A MICROCOMPUTER-BASED NETWORK OPTIMIZATION PACKAGE

Richard Henry Duff

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROCOMPUTER-BASED NETWORK OPTIMIZATION PACKAGE

by

Richard Henry Duff

September 1981

Advisor: G. G. Brown

Approved for public release, distribution unlimited

T200643

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): A Microcomputer-Based Network Optimization Package	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, September 1981 / 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Richard Henry Duff	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: September 1981 / 13. NUMBER OF PAGES: 130
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Networks, microcomputer, optimization, linear programming, nonlinear programming, mixed integer programming, minimum cost network flow, mathematical programming assignment model, transportation model, transshipment model, fixed charge network, nonlinear network.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An important branch of mathematical programming is concerned with optimization in systems described by networks. This paper describes an integrated suite of advanced techniques for dealing with minimum cost network flow formulations. Written in Pascal and implemented on a microcomputer representative of current small computer technology (the APPLE II), this package places unprecedented modeling versatility and solution capability on the analyst's desktop. Able

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 68 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

to solve small to medium size problems (3000 arcs or less) at reasonable speeds, programs to handle capacitated linear, nonlinear (convex separable), mixed integer and elastic ranged linear models in addition to comprehensive control and data management routines are included. Problem size and solution speed benchmarks are given for a variety of models.

Approved for public release, distribution unlimited

A Microcomputer-Based Network Optimization Package

by

Richard Henry Duff
Major, United States Marine Corps
B.S., Drexel Institute of Technology, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September, 1981

Thesis
D78365
c.1

ABSTRACT

An important branch of mathematical programming is concerned with optimization in systems described by networks. This paper describes an integrated suite of advanced techniques for dealing with minimum cost network flow formulations. Written in Pascal and implemented on a microcomputer representative of current small computer technology (the APPLE II), this package places unprecedented modeling versatility and solution capability on the analyst's desktop. Able to solve small to medium size problems (3000 arcs or less) at reasonable speeds, programs to handle capacitated linear, nonlinear (convex separable), mixed integer and elastic ranged linear models in addition to comprehensive control and data management routines are included. Problem size and solution speed benchmarks are given for a variety of models.

TABLE OF CONTENTS

I. INTRODUCTION 14

II. DESIGN AND IMPLEMENTATION CONSIDERATIONS 19

A. BACKGROUND 19

B. HARDWARE 20

1. Microcomputer Characteristics 20

2. Target Microcomputer 21

3. Machine Independence 22

C. MODEL AND SOLUTION TECHNIQUE SELECTION 22

1. Models 22

2. Solution Methods 23

D. PROGRAMMING LANGUAGE 26

1. BASIC 26

2. FORTRAN 27

3. Pascal 27

a. Standard Pascal 27

b. UCSD Pascal 29

c. Limitations of Pascal 31

E. MEMORY MANAGEMENT 33

F. THE USER INTERFACE 34

1. Data Input 34

2. Program Control 35

3. Programming Tactics 36

III. ALGORITHMS 37
A. GNET 37
1. Primal Revised Simplex Algorithm 37
2. Network Specializations 40
B. NLPNET 43
1. Basic Properties of Solution and Methods for Nonlinear Optimization 43
2. Newton's Method for Unconstrained Optimization 45
3. Truncated-Newton Methods for Unconstrained Optimization 47
4. Truncated-Newton Methods for Linearly Constrained Optimization 53
a. The "Reduced Problem" 54
b. The Search Direction 55
5. Primal Truncated-Newton Method Specialized for Networks 58
C. ENET 62
1. Ranged and Bounded Model 63
2. Elastic Model 64
3. Elastic Primal Revised Simplex Specialized for Networks 67
D. UNET 70
IV. PACKAGE DESCRIPTION 76
A. OVERVIEW 76
1. Package Components 76
2. Data Files 78

a. Structure 78
b. File Naming Conventions 80
3. User-Friendly Features 80
B. CONTROL AND UTILITY PROGRAM DESCRIPTIONS 82
1. Master Program 82
2. EDITOR Program 86
a. File Creation 88
b. File Alteration 89
c. File Transfer 89
3. Preprocessor Programs 89
C. SOLUTION PROGRAMS 90
1. GNET 91
a. Input Requirements 91
b. Internal Data Structure 92
c. Solution Segment 93
2. NLPNET 93
a. Input Data Requirements 93
b. Internal Data Structure 97
c. User Definable Parameters 97
d. Reports 100
3. ELASTICNET 100
a. Input Data Requirements 101
b. Internal Data Structure 102
c. Model Alteration 105
d. Reports 105
V. COMPUTATIONAL RESULTS 106

A. MAXIMUM PROBLEM SIZE 106
B. SOLUTION TIMES 109
C. MODELING FLEXIBILITY 111
VI. CONCLUSIONS AND RECOMMENDATIONS 118
APPENDIX A PACKAGE EXTERNAL DATA STRUCTURE (DATA FILES) 120
APPENDIX B TYPICAL REPORTS 121
LIST OF REFERENCES 127
INITIAL DISTRIBUTION LIST. 130

LIST OF TABLES

1. NOMINAL PACKAGE DATA PARTITION 108
2. PACKAGE MEMORY USAGE 108
3. ALLOWABLE ARC/NODE PARTITIONS 110
4. TYPICAL SOLUTION TIMES 112
5. OPTIMAL SOLUTION COMPARISON 113
6. THREE ECHELON DISTRIBUTION MODEL SOLUTION TIMES 117

LIST OF FIGURES

1. Development Hardware 21
2. Pascal Block Structure 28
3. Relationship Between P-Code and Pascal 31
4. Characteristics of the Newton Method 48
5. Truncated-Newton Method 50
6. Truncated Conjugate-Gradient Algorithm 51
7. An Elastic Ranged Constraint 65
8. A Rigid Ranged Constraint 65
9. Classical Equality Constraint 66
10. Relationship Between Logical Variables for ERLNP 67
11. Logical Arcs Generated for Node i 68
12. Primitive Enumeration Restrictions 73
13. Generic Enumeration Tree (Fixed Order) 75
14 Package Block Diagram 77
15. Typical Option Selection Menu 81
16. Typical Parameter Control Menu 82
17. Outer Command Logic 83
18. Solution Process Logic 84
19. Linearization Options 86
20. EDITOR Block Diagram 87
21. Generic Solution Program Segment Map 90
22. GNET Input Data Requirements 92

23. GNET Basis Representation 94
24. GNET Data Structure 95
25. NLPNET Input Data Requirements 96
26. NLPNET Basis Representation 98
27. NLPNET Data Structure 99
28. NLPNET Default Modification Menu 100
29. ELASTICNET Input Data Requirements 102
30. ENET Basis Representation 103
31. ENET Data Structure 104
32. UNET Data Structure 105
33. Three Echelon Distribution Model With Handling Costs 114
34. Modeling Fixed Costs 116

NOTATION

Except as otherwise indicated in the text, the following notational conventions have been used for all mathematical expressions:

Scalars	Lower case Greek and Latin letters (e.g., α, a).
Vectors	Lower case Latin letters with a bar above ($\bar{a}$).
Vector Components	Lower case Latin letters with a lower case Latin subscript (a_i).
Matrices	Upper case Latin letters (A).
Matrix element	Lower case Latin letters with lower case Latin subscripts (a_{ij}).
Transpose	Superscript T (A^T).
Sequences	Enclosed in braces ($\{a_k\}$).

ACKNOWLEDGEMENT

I am grateful to Professor Ron S. Dembo of Yale University for his assistance rendered in the implementation of NLPNET. Additionally I would like to express my appreciation to Professors Victor A. Cabot of Indiana University and Richard E. Rosenthal of the University of Tennessee for their academic support and many valuable suggestions. Although thesis advisors are normally implicitly recognized, I feel that such acknowledgement is inadequate in this case. Professors Brown and Washburn have expended great effort to ensure that this project was a worthwhile learning experience in every respect. To both of these gentlemen, a sincere thank you. Finally, I wish to thank my wife Cheryl for her patience and support.

I. INTRODUCTION

Mathematical programming can be defined as the use of mathematical representations (models) to plan (program) an allocation of scarce resources among competing activities [Ref. 1]. An important branch of mathematical programming deals with optimization in systems described by networks or collections of points (nodes) connected by links (arcs). Such models arise explicitly in a variety of applications and include many familiar distribution and transportation problems. Bradley [Ref. 2] suggests that network models have been so widely used because: (1) they accurately model many applications, (2) they are more readily accepted by nonanalysts than other models (they pictorially resemble the physical process being modeled), and (3) efficient algorithms are available. Additionally, many models can easily be transformed into equivalent network representations by direct manipulation (e.g., assignment problems) or exploitation of a primal-dual relationship (e.g., critical path problems). The object of all these formulations is usually to minimize the cost of moving a single commodity through the network. The general form of this problem, the network programming problem (NPP), is

$$
\begin{array}{llll}
\text{(NPP)} & \text{MINIMIZE} & f(\bar{x}) & \text{(cost function)} \\
 & \text{s.t.} & A\bar{x} = \bar{b} & \text{(node flow-conservation constraints)} \\
 & & \bar{l} \leq \bar{x} \leq \bar{u} & \text{(arc flow bounds),}
\end{array}
$$

where $\bar{x}$ is an n-dimensional vector

(n is the number of arcs),

x_i represents the magnitude of the flow on arc i,

A is an m by n (m is the number of nodes)

matrix defined as follows:

$$a_{ij} = \begin{cases} +1 \text{ if arc } j \text{ is directed away from node } i \\ -1 \text{ if arc } j \text{ is directed toward node } i \\ 0 \text{ otherwise,} \end{cases}$$

$\bar{b}$ is an m-dimensional vector of flow requirements at

each node, and

$f(\bar{x})$ is some function that relates cost to arc flow.

Although NPP can be solved by classical general purpose constrained methods (depending on the exact form of $f(\bar{x})$), better techniques exist. Specialized data structures employed in conjuction with modifications of traditional optimization procedures have resulted in the development of extremely efficient algorithms for the solution of NPP [e.g., Ref. 3]. This work has been motivated in part by the intrinsic usefulness and wide applicability of these models and further stimulated by the increased availability of computational devices. Due to these advances, the operations analyst is now able to represent larger network models, and answer questions concerning their optimal flow easier than ever before.

As interest in network programming continues to grow, the supply of applications software will undoubtedly keep pace. This has been the case to date; however, the vast majority of the emerging software appears very specific in nature and tailored to particular classes of network models. Certainly there are many superb codes available, but if, for

example, the capability to work with both linear and nonlinear cost functions is desired, then the employment of two separate and possibly quite different programs is required. Even a relatively mundane task such as linearizing a nonlinear model to obtain a feasible starting solution point either requires incorporation of extra code or an exercise in data manipulation and program linking. Such shortcomings indicate that unified network flow programming packages able to cope with several different types of models would be quite useful.

Concurrent with these advances in network programming, computer technology has experienced breath-taking progress. Not only have computers improved in sheer power, but they have become increasingly compact and less expensive (relative to capability). A dramatic example of this data-processing revolution is the evolution of desk-top computers. These small computers, the so-called microcomputers, appeared on the scene in the early 1970's and today can provide computer power comparable to some room-sized machines of a decade ago at a tiny fraction of the cost. More importantly, these devices are becoming so common that their accessibility to technical personnel is forecasted to be virtually universal in the near future [Ref. 4]. E. M. L. Beale [Ref. 5] commented on this dissemination of computer power and predicted increased use of microcomputers by operations analysts to solve smaller models locally with reliance on computer centers for large projects. Tanenbaum [Ref. 6] suggests that the cost of small computers relative to communication expenses now makes it attractive to analyze data at its source and send only summaries back to large computers via networking arrangements.

Even though the hardware is available and its potential clearly recognized, suprisingly little operations analysis software of reasonable complexity and generality has been reported for microcomputers. Aside from a few decision analysis programs [e.g., Ref. 7, 8, 9], effort has been mainly concentrated on statistical applications. Morgenson [Ref. 10] and Isbell [Ref. 11], for example, have developed extensive data analysis packages incorporating sophisticated techniques. Additionally, there are numerous commercial statistical products of varying quality and capability for microcomputers.

Surprisingly, microcomputer-based mathematical programming software is still virtually non-existant, despite the obvious utility of such programs. Undoubtedly, there are many ad hoc, rudimentary implementations of basic methods; however, no reference to any work of consequence has been found in the open literature. Feasibility studies [Ref. 12; 13] verify the desirability of optimization packages for small computers, but only explore rather elementary network algorithms. These studies further suggest that network flow problems are likely candidates for microcomputer solution because of the efficient algorithms at hand. Economic justification for employing the microcomputer for optimization purposes has not been established conclusively. One attempt to favorably compare such use with the alternative of large, general purpose computer systems [Ref. 13] has been severely criticized [Ref. 14] and the issue remains undecided.

The research reported here investigates the construction of unified network flow optimization packages, the mathematical programming potential of microcomputers, and (secondarily) the economic feasibility of such

devices with respect to optimization applications. This work was undertaken with the following (superficially disparate) goals:

1. Development of a unified, versatile network-flow optimization package capable of handling a variety of models and utilizing "state-of-the-art" algorithms.
2. Implementation of this package on a widely available microcomputer to explore the usefulness of smaller computers in an optimization context.

Portions of this work were presented at the CORS/ORSA/TIMS joint meeting in Toronto, May, 1981. At that international meeting, attendees expressed surprise that such an integrated network optimization package used a microcomputer as a host.

II. DESIGN AND IMPLEMENTATION CONSIDERATIONS

Microcomputer implementation of large programming projects requires careful consideration of every aspect of program design in order to fully exploit the limited resources available. This section briefly discusses the design criteria utilized and outlines the rationale for the decisions that shaped the package.

A. BACKGROUND

Invariably many different program configurations can be constructed to accomplish a given task, however, some approaches are better than others. Program effectiveness is a well-studied field and the literature contains many characterizations of superior software designs. Kreitzberg and Shneiderman [Ref. 15], for instance, define a "good" program as one that is correct (provides desired results), fast, accurate, hardware independent, efficient with storage, and easily modified. Thenson [Ref. 16] and Lientz [Ref. 17] further suggest that operations analysis software should place heavy emphasis on "user impact" or ease of use. These attributes, commendable as they may be, are of little value if the program cannot solve meaningful problems in a reasonable amount of time. Meaningful, in this context, includes not only the intrinsic usefulness and generality of the modeling facilities provided, but also the size and complexity of the representable formulations. Since it is highly unlikely that a program can be optimal in every respect, compromises are inevitable.

Numerous factors interact to establish program limitations with the inherent capabilities of the host computer playing the dominant role. Choice of algorithms, data structures, programming language, memory and peripheral management, and programming tactics are also significant determinants of ultimate software efficiency. For the traditional user of large computer systems, the improper choice of one or more of these variables usually results in merely a slightly degraded package rather than outright failure. The small-computer environment is less forgiving and does not allow this luxury without severe performance penalties.

B. HARDWARE

1. Microcomputer Characteristics

A typical microcomputer system consists of a central processing unit, memory, peripheral devices, and software. Exclusive of peripherals, such packages physically resemble an electric typewriter. The specific form of the processing unit varies, but most can directly address 65,536 eight-bit words of random-access memory. This "fast" (typically 200-400 nsec) or core memory is usually supplemented by slower (but more plentiful) storage (worst case, 1-2 seconds seek time) in the form of disk drives. Each disk drive provides approximately one hundred thousand (or more) words of memory space on removable magnetic media. The number of disk drives allowed, their individual capacities, and access times depend on the microcomputer. More sophisticated (and expensive) offline memory devices are also available, capable of storing millions of words. Common peripherals include communication equipment to access other computers, video displays, and printers.

2. Target Microcomputer

Clearly, as technology is changing so rapidly, any attempt to identify an optimal hardware selection would be futile. Instead, an acceptable system representative of available products, the APPLE II[1] microcomputer, is considered. All software development has been performed on an APPLE II system configured as shown in Figure 1.

---

1. APPLE II microcomputer with 65,000 words of memory.
2. Two disk-drive offline storage devices (150,000 words each).
3. UCSD Pascal (language card).
4. Communications device (modem).
5. Printer.
6. Eighty character by twenty-four line video display.

Fig. 1. Development Hardware

Although this machine is not the most powerful of its class, it is a reasonable choice for many operation analysis applications due to its availability, capability, and low cost. Introduced in 1977, the APPLE II is a widely-used device, still in production, and likely to remain so for the next few years [Ref. 18]. As over 200,000 units have been manufactured and distributed worldwide, a broad base of technical support exists and numerous firms offer peripheral products. The APPLE II can support all of the popular microcomputer programming languages (BASIC, Pascal, FORTRAN, and Assembly language). Easily expandable,

---

[1]APPLE II is a trademark of Apple Computer Inc.

complete systems can be purchased for between two and four thousand dollars.

3. Machine Independence

All computers, regardless of size, have unique features and one must be careful if machine independence of software is desired. Given the variety of microcomputers available today and the rapid changes expected, portability between existing and conceptual machines is necessary if the considerable investment involved in the development of sophisticated programs is to be protected. Thus the importance of machine independence should not be underestimated. For this reason, features peculiar to the APPLE II have been avoided wherever possible. Thus the software presented should run, with minimal modifications, on any equivalent hardware package.

C. MODEL AND SOLUTION TECHNIQUE SELECTION

The determination of specific processes to model and subsequent selection of solution techniques compatible with computing resources are critical issues in the design of any optimization software.

1. Models

Once the general class of problems has been established, a few carefully chosen models often can adequately represent the majority of anticipated situations. Specialized models can then be added as necessary. The following minimum cost network flow models are considered essential for any comprehensive package:

- Linear cost function with bounded variables.
- Nonlinear convex separable cost function with bounded variables.

As useful as these two models are, they cannot depict some desired formulations. Additional flexibility is provided by:

- o Linear cost function with bounded variables and elastic ranged constraints (see Section III, Subsection C).
- o Linear cost function with elastic range constraints and bounded variables, any of which may also be specified as "l-u" variables (only allowable flow is at one of the bounds).

These four models comprise a basic package capable of representing a wide variety of single commodity network flow problems. Included are fundamental examples of linear, nonlinear, and mixed-integer network optimization models. Building from this network paradigm, these features can be used to represent myriad mathematical models. Other models could easily be added, however, time constraints precluded further extension of the package.

2. Solution Methods

When dealing with microcomputers, algorithms are difficult to select. Simple algorithms require little storage, but are typically inefficient and slow. More complicated approaches offer speed at the expense of increased memory usage and expanded data structure requirements. Large computers are able to use the efficient algorithms because the additional storage requirements are infinitesimal compared to the total memory available. For small machines, the choice is not so clear. A large portion of memory is consumed by the added complexity of the advanced algorithms, resulting in significant reduction of maximum problem size. If one chooses the simpler methods, larger programs are accommodated but processing time can soar to unacceptable levels for all problems.

Execution speed appears to be far more important than maximum problem size. Providing quick answers to small problems is the most likely optimization role for microcomputers at the present time. Very large problems will continue to require extensive computational facilities well beyond the capabilities of current microcomputers. Given the predicted advances in small computers, memory-saving techniques should decrease in importance as devices with greater storage capacities become available. Using inefficient methods instead of the more capable but memory-intensive algorithms does not seem to be justified.

Numerical representation must also be considered. Small computers are usually more restricted than their larger counterparts in (1) mantissa precision, (2) real variable exponent range, and (3) maximum allowable integer size. If provisions exist (or can be created) to deal with these problems, performance degradation may result. Thus candidate algorithms for microcomputer use must be relatively insensitive to such limitations.

The programs selected for these models are considered state-of-the-art and all represent current research efforts. Utilizing advanced algorithms and efficient data structures, these programs have been included on the basis of their suitability for microcomputer adaptation and mutual compatability. The original programs were coded in FORTRAN for implementation on large computers. Each program has been translated into UCSD Pascal and modified as necessary to enhance its efficiency in a microcomputer environment.

GNET [Ref. 3] is the foundation of the package. A highly regarded and widely-used code, it is one of the fastest methods available for solving linear minimum cost network models with bounded variables.

Additionally, GNET can be used to linearize nonlinear problems and produce initial feasible solutions. The documented speed of the method [Ref. 2] and its efficient data structures make it ideal for microcomputer implementation.

Nonlinear formulations are solved by NLPNET [Ref. 19], a program by Dembo that utilizes data structures in the spirit of GNET and new adaptable direction-finding techniques. The user of NLPNET can control the accuracy of the direction-finding process and thus the amount of computation effort expended for a solution. This feature makes such an algorithm perfect for microcomputers.

Elastic ranged constraints are handled by ENET, a recently introduced code by Brown and Graves [Ref. 20]. An extension of GNET, it uses the same efficient methods and data structures.

Finally, UNET, another new code by Brown and Graves [Ref. 20], is used to solve elastic ranged constraint problems with bounded variables, some of which may be specified as "l-u" variables (flow at either bound, only). Also employing data structures similar to GNET, this algorithm requires successive calls to ENET to solve enumeration subproblems.

Although ENET can duplicate the capabilities of GNET, the latter is explicitly retained to solve larger problems of the simpler structure amenable to GNET. This is not an algorithmic disadvantage of ENET, but a consequence of the package design. Since UNET repeatedly calls ENET, both procedures are kept in memory together to reduce disk access time. Therefore, as the programs are constructed, GNET is a more compact code than the ENET/UNET combination. It is intended that future revisions of the package will employ only ENET and UNET.

D. PROGRAMMING LANGUAGE

Pascal, specifically the University of California at San Diego (UCSD) version, appears to currently offer the best programming language facilities for large-scale microcomputer programming projects. Possible alternatives are BASIC and FORTRAN, languages that suffer from crippling shortcomings as presently implemented on microcomputers.

1. BASIC

Once the only high-level language available for small computers, BASIC remains very popular. While BASIC is perhaps suitable for some applications, serious mathematical programming efforts are hampered by its limited speed, primitive programming power, and lack of transportability. When installed as an interpreted language (the usual case), each source statement must be translated into machine language every time it is encountered in the logical flow of the program. This results in long execution times, especially for complex programs with many iterative structures (i.e., optimization programs). Additionally, most dialects of BASIC allow only global (accessable to all program portions) variables with restrictive naming conventions. Variable name conflicts often arise in such circumstances greatly reducing the portability of subprograms. The use of subprograms is further hindered by the inability to pass parameters between program fragments without resorting to global variable reassignments. Compiled versions (one-time translations of source statements to machine language) of BASIC are available that supposedly alleviate these problems, however, they are necessarily machine-specific, further aggravating the transportability issue. There are probably as many different versions of BASIC on the market as there are types of

microcomputers. Hardware manufacturers modify the language to fit their particular needs resulting in potential difficulties when programs are moved to different machines. It is indeed a rare BASIC program that can be transported without modification, with the chances of such success diminishing as program length and complexity increases.

2. FORTRAN

Implementations of FORTRAN for microcomputers, although increasing in number, are still rather uncommon. A compiled language, FORTRAN compares favorably to other languages in terms of speed and programming power. The biggest drawback of FORTRAN is lack of standardization, a hinderance that has plagued FORTRAN users on large computers for years. Compiled code is completely machine-specific and thus not portable. Transportability of the source statements varies depending of the degree of similarity between the two versions of FORTRAN in question. This lack of standardization combined with the inability to exchange compiled code between different microcomputers makes FORTRAN a poor choice if program portability is a goal.

3. Pascal

a. Standard Pascal

Pascal is a relatively new language (compared to BASIC and FORTRAN) having been formally defined by N. Wirth in 1971 [Ref. 21]. Named after the famous mathematician Blaise Pascal, the language was originally intended as a vehicle to teach computer programming. An excellent instructional tool, Pascal is now recognized as a powerful general-purpose language. Similar to ALGOL, Pascal provides a rich set of program and data structuring tools. Thus a great portion of

the housekeeping tedium of programming is assumed by the language itself (in the form of the compilation process) and is totally transparent to the user. Some of these facilities can be simulated in other languages through skillful programming, but this adds complexity to the development process. Pascal, by admitting long variable names and free-form coding, is a self-documenting language further reducing development and maintenance effort. Totally modular in concept, programs consist of blocks of similarly structured code with true local and global variables allowed at all levels. A Pascal program is therefore a logical collection of independent modules that allows direct translation of complicated algorithms into source code by partitioning the problem into smaller and more manageable pieces. This divide and conquer technique, commonly known as "structured programming," is an effective method to address large projects. Figure 2 is an example of this block structure. Extension of Pascal programs is a simple matter of adding modules of code and linking them to the original program via parameters. Pascal also requires explicit type definition of all variables, thus minimizing the chance of a non-local typographical mistake introducing a program error.

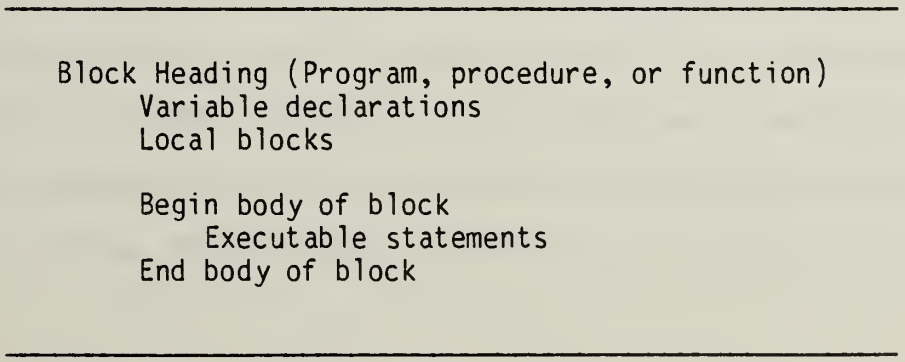


Fig. 2. Pascal Block Structure

b. UCSD Pascal

These impressive programming features alone are enough to warrant the use of Pascal for difficult microcomputer programming projects. When one examines the UCSD version, however, the evidence seems overwhelming. An extension of standard Pascal, UCSD Pascal[2] was developed by a group headed by K. L. Bowles in 1975. Explicitly designed to run in a small computer environment, it retains most of the features of standard Pascal while adding many extensions that increase the power of the language. For our applications, we are interested in two of these additional facilities that (1) enhance memory manipulation capabilities and (2) provide for extremely transportable programs.

The memory management techniques offered by UCSD Pascal are truly remarkable for a microcomputer software system. The programmer has complete control over which portions of code are to be in memory at a given point in the program execution. For small programs, the entire code is generally loaded into memory at the beginning of execution while larger programs can be segmented to provide the desired partitions. A small portion of the program, again totally definable by the programmer, must be in residence at all times to control the overlaying process. Complete programs can be called into memory, each with its own local memory control, executed, and other programs subsequently activated. In this manner, programs can be chained together to form coherent packages. Additionally, groups of often-used blocks of code can be pre-compiled and

---

[2]UCSD Pascal is a trademark of the Regents of the University of California.

placed in libraries. When activated, the specific library code is loaded into memory with the calling program and executed. Library routines can either remain in memory for the entire program execution process or be discarded after each use (at the programmer's option). These methods allow programs that would otherwise be impossible to implement on a microcomputer--the package presented is such a case as the combined code of the component programs far exceeds the memory capacity of most microcomputers.

UCSD Pascal is a hybrid compiled/interpreted language, a concept that produces unprecedented machine independence. Source statements are first compiled to an intermediate pseudo (p) code that is then executed by a machine-dependent interpreter. This greatly enhances portability as the interpreter, which normally takes about six man-months to write [Ref. 22], is the only part of the system that must be changed to take advantage of a new hardware configuration. The original compiled source code can be transported without modification to any machine that has a p-code interpreter installed. Figure 3 depicts the relationship between the components of the p-code system (Lewis [Ref. 23]). Such interpreters now exist for all the commonly used microcomputer central processing units [Ref. 24]. The use of p-code is not restricted to UCSD Pascal; other languages (e.g., FORTRAN-77 and PL/I) have been treated in this manner to achieve similar portability. However, UCSD Pascal is currently the only p-code implementation that is widely available.

As previously indicated, speed is an important programming language attribute. The hybrid nature of UCSD Pascal results in

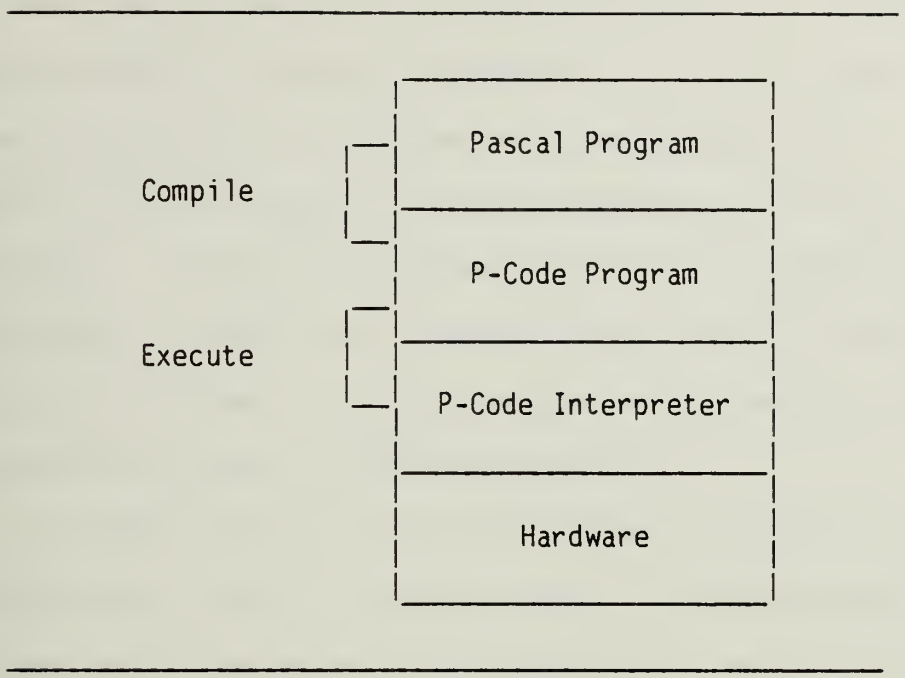


Fig. 3. Relationship Between P-Code and Pascal

execution time somewhat slower than pure-compiled languages such as FORTRAN, but much faster than interpreted languages. Gagne [Ref. 25] reports that Pascal runs three times as fast as the very best BASIC and ten to thirty times faster than most. Informal timing tests comparing Pascal with APPLE II BASIC confirms these performance assertions. Any loss of speed through the interpretation of p-code is more than offset by the other desirable features of the language, especially portability.

c. Limitations of Pascal

Pascal is not without its difficulties--some are minor irritants while others are more important. Foremost is the precision of real variables allowed by the APPLE II version of UCSD Pascal. Without taking any special programming measures, six to seven significant figures can be accommodated in a mantissa representation equivalent to IBM 360/370

single precision. It must be understood that this is an implementation issue on the APPLE II and not an intrinsic limitation of Pascal. By suitable programming, the precision can be extended; however, it was decided not to do so because of the speed degradation that would surely have resulted. The only package module that utilizes vast amounts of real arithmetic is NLPNET, the nonlinear network code. As shall be shown, for the nonlinear problems that have been attempted, limited precision does not appear to hinder performance.

Another disturbing omission is that all arrays are static and created with a size and type dope vector. Linkage conventions require that actual parameters in subprogram calls and formal parameters in the routine called must agree in type and dimension. This precludes problem-dependent memory management as an automatic feature. Since array bounds must be predeclared, their alteration, for any reason, requires complete recompilation.

Also, compilation itself is a rather slow process. The nominal compilation rate for APPLE II UCSD Pascal is two hundred source statements per minute. This becomes tedious for the long programs developed for this package, some of which are on the order of three thousand source lines (exclusive of comments).

The final important criticism is the lack of a predefined exponentiation operator, a crucial primitive for nonlinear optimization work. For development purposes, an exponentiation operator has been written in UCSD Pascal. A much better solution would be a fast machine language function, or utilization of special arithmetic processing hardware.

E. MEMORY MANAGEMENT

A major challenge to the microcomputer programmer attempting to code a complicated algorithm is memory management. The interaction between the two types of memory (fast "core" and slower peripherial) is a delicate affair and an extremely important influence on overall program efficiency. As a general rule, optimization programs will always expand to fill available core memory. Memory not occupied by program code will contain the data structure representation. In order to extend problem size further, either the data or the program code (perhaps both) must be partitioned in some manner and the resulting pieces moved in and out of fast memory as needed. Both these approaches are feasible with today's microcomputers, but the intrinsic code management features of UCSD Pascal make program code segmenting much easier to implement.

Data partitioning (so-called "in-core/out-of-core" operations) requires additional control logic and data structures to work effectively. As the purpose of this research is demonstrative in nature, data partitioning experimentation has been deferred for future package enhancement. Only program code segmentation has been utilized to dynamically manage memory resources. Such a process is termed "swapping" or "overlaying."

Due to the iterative nature of mathematical programming algorithms and the time expense involved in accessing current disk drives, one must be extremely mindful of how the program is segmented. It is desirable to have minimal program code in residence at all times during the critical solution process. If the divisions are too fine, then execution speed suffers as the disk is constantly being accessed. Keeping too much of the program in core reduces disk access at the expense of data storage.

Therefore, each program must be examined carefully and partitioned to yield the best compromise of speed and storage management.

F. THE USER INTERFACE

All computer programs require a certain level of human interaction to produce results. Depending on the particular application, this participation may range from minimal operations such as data input and output interpretation to complex control decisions. Whatever the degree of interaction necessary, software must be designed with the user in mind if maximum benefit is to be derived from its use. Programs that properly take the human element into consideration and attempt to promote meaningful man-machine dialog are termed "user-friendly." As microcomputers are interactive devices usually operated by a single person with the entire computing system at arm's reach, user involvement is inescapable. Software designed for the small computer must therefore be user-friendly.

1. Data Input

Thenson [Ref. 16] states that programs should be designed to minimize the probability of errors in the user input process. Likely causes of these errors are incompatible input format, invalid characters in the input field, and inadmissible input values.

Input formats should be flexible whenever possible, allowing user control to effect format reconfiguration. Otherwise, as a default, the easiest method for the user should prevail and internal conversion can then be undertaken as needed.

The frustrating problem of invalid characters in the input stream can be avoided by the use of buffering and filtering techniques to isolate undesired values. This error is often encountered during

keyboard input where, for example, an alphabetic character is entered when a numeric digit is required. Most operating systems respond to this difficulty by terminating program execution, an intolerable situation. Morgenson [Ref. 10] describes a procedure in detail where all numeric values are input as string variables. A lexicographic scan on the string is then performed to identify offending characters and construct the number for program use. We use a modification of his technique.

Inadmissible values can be isolated by screening all input prior to its use. Preferably, this is performed as the data is received by the program. Simple precautions such as confirming that the data conforms to required numerical ranges (range screening) and sign restrictions can prevent unexplained abnormal program terminations.

Every attempt should be made to not only detect errors, but also to inform the user of their presence. This involves either providing facilities to correct the error and continue if possible, or executing a graceful program exit in the event of fatal errors. Meaningful error messages must be displayed in order that proper corrective action can be taken.

2. Program Control

User-friendly techniques can also be easily applied to interactive control during program execution. A popular method to do this involves menu-driven selections where various choices are displayed along with simple commands. The user selects a command thereby activating either the desired program segment or additional sub-menus as appropriate. By using single keystroke commands, combined with range screening and

suitable error messages, a very large number of alternatives can quickly be considered.

Displays should be designed so that the user has enough information to make decisions without being overburdened visually by extraneous material. Selections should be arranged with the most frequently used choices the easiest to invoke. A good example of this is providing access to default parameters only on demand. These values are then transparent to the user, yet readily available if changes are to be made.

3. Programming Tactics

The addition of user-friendly features invariably results in more complicated code and longer development time. This is a small price to pay in view of the utility gained. Programs that are difficult to use typically go unused and often must be changed, at far greater cost than an equivalent original design, to instill user confidence. If user-friendly facilities are incorporated at the conceptual phases of software development and sensibly blended into the program structure, their cost can be minimized.

For optimization programs, there is a clear delineation between interface and solution modules. Even with time-shared, or dedicated microcomputer systems, the code supporting user-friendly facilities can be easily isolated from the iterative portions of the program. In this manner, the programmer can take advantage of these indispensable input/output techniques without impairing the solution process. With proper memory management discipline, the only penalty is increased secondary (offline) storage consumption.

III. ALGORITHMS

The solution programs chosen for inclusion in this package represent advanced methods for solving minimum cost network flow problems. Detailed descriptions of the algorithms employed are scattered among various references, and are not available in a collected form. This section outlines the fundamental ideas underlying each algorithm to provide a single source document. The discussions are necessarily brief and the reader is directed to the primary references for in-depth descriptions.

A. GNET

GNET is an extremely efficient and elegant code that solves network flow problems with linear costs and bounded variables. GNET uses the well-known primal revised simplex algorithm specialized for networks.

1. Primal Revised Simplex Algorithm

Consider the following linear program:

$$\text{(LP)} \quad \text{MIN} \quad \bar{c}^T\bar{x}$$

$$\text{s.t.} \quad A\bar{x} = \bar{b}$$

$$\bar{0} \leq \bar{x} \leq \bar{u}$$

where $\bar{c}$ is a vector of cost coefficients and A is a matrix of technological coefficients. Any lower bounds on the variables, $\bar{x}$, have been eliminated by transformation. The matrix A may be partitioned into two sub-matrices B and N to yield

$$A = [B : N]$$

where a matrix of linearly independent columns, a basis, is represented by B and N consists of the remaining columns of A.

The vectors $\bar{c}$ and $\bar{x}$ may be similarly partitioned to form

$$\bar{c}^T = (\bar{c}_B \; \bar{c}_N) \; ,$$

$$\bar{x}^T = (\bar{x}_B \; \bar{x}_N) \; ,$$

which implies that

$$\bar{c}^T\bar{x} = \bar{c}^T\bar{x}_B + \bar{c}^T\bar{x}_N \; .$$

Redefining variables so that every nonbasic variable is at its lower bound simplifies the procedure. Utilizing the notation of Bradley, Brown and Graves [Ref. 3], let x_k be replaced by $u_k - x_k$ (reflection) whenever x_k reaches its upper bound.

Given a feasible basis, there exists a unique solution $\bar{x}$, such that $B\bar{x} = \bar{b}$, and a basic solution where $\bar{x}^T = (\bar{x}_B \; \bar{0})$.

Any solution satisfying the constraints can be written as

$$A\bar{x} = [B \; N](\bar{x}_B \; \bar{x}_N)^T = B\bar{x}_B + N\bar{x}_N = \bar{b} \; ,$$

or

$$\bar{x}_B = B^{-1}\bar{b} - B^{-1}N\bar{x}_N \; .$$

Denoting $B^{-1}N$ as Z yields

$$\bar{x}_B = B^{-1}\bar{b} - Z\bar{x}_N \; .$$

The value of the objective function $\bar{c}^T\bar{x}$, expressed in terms of $\bar{x}_N$ is then $\bar{c}^T\bar{x} = \bar{c}_B^TB^{-1}\bar{b} + (\bar{c}_N^T - \bar{c}_B^TB^{-1}N)\bar{x}_N$.

Differentiating with respect to $\bar{x}_N$ provides the rate of change of the objective function in response to changes in $\bar{x}_N$:

$$\frac{\partial \bar{c}^T\bar{x}}{\partial \bar{x}_N} = \bar{c}_N^T - \bar{c}_B^TB^{-1}N \; .$$

The vector $\bar{c}_B^TB^{-1}$ is the solution to

$$\bar{w}^TB = \bar{c}_B^T \; , \qquad (1)$$

where $\bar{w}$ is commonly referred to as the dual solution vector.

Favorable movement of the objective function (minimization) from a feasible solution is indicated by

$$c_j - w_j N^j < 0 \qquad (2)$$

for nonbasic variables j at zero flow. Any nonbasic variable x_j satisfying (2) will induce the following change in the solution (assuming all other nonbasic variables remain fixed):

$$\bar{x}_B = B^{-1}\bar{b} - Z^j x_j \qquad (3)$$

$$\bar{x}_N = (0,\ldots,x_j,\ldots\ 0)\ . \qquad (4)$$

As this solution satisfies the explicit constraints, enforcement of the bound constraints will ensure that feasibility is maintained. Updating the variable partition completes the procedure.

Bradley, Brown, and Graves [Ref. 3] give the following interpretation of the revised simplex procedure:

STEP 0: Obtain a feasible solution.

REPEAT

STEP 1: Priceout. Select a candidate variable to enter the basis that satisfies (2).

STEP 2: Ratio. Find the greatest bound such that the incoming variable:

a. does not exceed its upper bound, or

b. drives a basic variable to its lower bound, or

c. drives a basic variable to its upper bound.

STEP 3: Pivot. Update the solution using (3) and (4). If case (a) of STEP 2 applies, reflect the candidate incoming variable chosen in STEP 1 and leave the basis and dual

solution unchanged. For case (b), change the basis (pivot) and find a new dual solution. Case (c) requires that the outgoing variable first be reflected and then a basis and dual solution update performed.

UNTIL STEP 1 fails to find a favorable candidate (optimality).

2. Network Specializations

Recall NPP[3]

$$\text{(NPP)} \quad \text{MINIMIZE} \quad f(\bar{x})$$
$$\text{s.t.} \quad A\bar{x} = \bar{b}$$
$$\bar{l} \leq \bar{x} \leq \bar{u}$$

where A is a node-arc incidence matrix. Restricting $f(\bar{x})$ to be a linear function yields a linear network programming problem (LNP):

$$\text{(LNP)} \quad \text{MINIMIZE} \quad \bar{c}^T\bar{x}$$
$$\text{s.t.} \quad A\bar{x} = \bar{b}$$
$$\bar{l} \leq \bar{x} \leq \bar{u}$$

Two well-known results that characterize the bases of LNP allow for an extremely efficient specialization of the primal simplex procedure to be applied to LNP. First, all bases for LNP are formed by a set of columns which correspond to a spanning tree for the graph represented by LNP. Additionally, any such basis matrix B can always be placed in triangular form by simple row and column permutations. The background pertaining to these results can be found in any elementary network programming reference (e.g., Kennington and Helgason, [Ref. 26]).

Triangulation of the basis matrix B implies that Z, which is defined as $B^{-1}N$ and thus the solution to $BZ = N$, can be obtained

---

[3]See Section I.

by direct solution (back substitution). Also, the dual variables $\bar{w}$ can be found by forward substitution of (1).

Characterization of network bases represented as spanning trees for the underlying graph suggests an easy method to perform the ratio test and pivot steps of the simplex method. Since a spanning tree by definition [Ref. 26] is a connected acyclic graph, any incoming nonbasic variable will form a cycle with the basis tree. Changes in flow as a result of an introduced arc will exclusively occur in unit amounts on the cycle in question. All off-cycle basic arcs will be unaffected. The direction of flow change (decrease or increase) will depend on the orientation of a particular basic arc with respect to the incoming arc. The basic arc on the cycle that reaches its bound first is removed from the basis (unless the incoming arc reaches its bound first).

Bradley, Brown, and Graves [Ref. 3] employ the following algorithm:

STEP 0: Given a feasible starting solution

REPEAT

STEP 1: Priceout. Given the dual variables $\bar{w}$, the priceout becomes $c_k - w_i + w_j$ for nonbasic arc k that is directed from node i to node j. Select an incoming arc k with $c_k - w_i + w_j < 0$ or terminate with the current solution as optimal.

STEP 2: Ratio Test. Three cases analgous to bounded variable simplex for which:

a. The incoming arc reaches its opposite bound u_k.

Otherwise a basic variable is selected as the outgoing variable (for basic arcs on cycle with incoming arc with opposite orientation).

b. Incoming arc drives basic variable down to its lower bound, (for basic arcs on cycle with incoming arc with same orientation).

c. Incoming arc drives basic variable up to its opposite bound.

STEP 3: Update (depending upon ratio test case result).

a. Reflect incoming arc.

b. Pivot update (see below).

c. Reflect outgoing arc, pivot update.

Pivot Update: simultaneously perform a one-pass update to:

1) modify flow on arcs in cycle by constant equal to ratio result,

2) update basis tree representation ("rehang"),

3) update dual variables for nodes whose precedessor chain to the root is changed by pivot operation (these are the rehung nodes).

UNTIL OPTIMAL.

The algorithm requires only integer arithmetic with most operations involving only addition and subtraction. Use of programming tactics and coding efficiencies described in Reference 3 results in an extremely efficient and compact code, ideal for microcomputer applications.

B. NLPNET

Minimum cost nonlinear network flow problems are solved utilizing NLPNET [Ref. 19], a primal approach based on controlled truncation of a conjugate-gradient method for solving the Newton equations. As such, NLPNET is an extension of the unconstrained Truncated Newton methods described in [Ref. 27]. Operating on a maximal basis in the manner of Dembo and Klincewicz's Scaled Reduced Gradient method [Ref. 28], NLPNET is designed to take advantage of the highly desirable local convergence properties afforded by Newton methods while avoiding the global convergence problems and computational overhead traditionally associated with these procedures. These methods belong to the class of "inexact Newton Methods" [Ref. 29].

1. Basic Properties of Solutions and Methods for Nonlinear Optimization

Consider the unconstrained nonlinear programming problem (NLP).

(NLP) MINIMIZE $f(\bar{x})$

$\bar{x} \in R^n$

where

$f: R^n \rightarrow R$ is a nonlinear function with the following properties.

a. f is continuous and twice differentiable.

b. At a relative minimum $\bar{x}^*$, the gradient $\bar{g}(\bar{x}^*)$ vanishes and the hessian matrix $H(\bar{x}^*)$ is positive definite.

c. For all $\bar{x}_0 \in R^n$, the level sets

$L(\bar{x}_0) \equiv [\bar{x} : f(\bar{x}) \leq f(\bar{x}_0)]$ are bounded

(e.g., Dembo and Steihaug, [Ref. 27]).

Algorithms to solve NLP vary considerably in specific approach, however, most are iterative descent methods. (Iterative means that the algorithms generate a series of points based on the preceeding points while descent implies that each new point produced by the algorithm improves the solution by reducing the value of the objective function.) In this manner, such methods would in the ideal case converge to a solution point of NLP.

For NLP, we must distinguish between two types of solution points: local minimum points and global minimum points.

Definition [Ref. 30]. A point $\bar{x}^* \in R^n$ is a local minimum point of NLP if there is some $\epsilon > 0$ such that $f(\bar{x}) \geq f(\bar{x}^*)$ for all $\bar{x} \in R^n$ within distance ϵ of $\bar{x}^*$.

Definition [Ref. 30]. A point $\bar{x}^* \in R^n$ is a global minimum point of NLP if $f(\bar{x}) \geq f(\bar{x}^*)$ for all $\bar{x} \in R$.

Similarly, the concept of convergence can be viewed in a global and local context. Global convergence deals with the actual determination of a solution point from an arbitrary starting point and is not to be confused with convergence to a global optimum.

Definition [Ref. 30]. An algorithm is said to be globally convergent if for arbitrary starting points, it is guaranteed to generate a series of points converging to a local solution point (local minimum).

Local convergence, on the other hand, refers to the properties of the algorithm in the vicinity of a solution. For example, asymptotic speed of convergence. One of the more important measures of this speed is order of convergence.

Definition [Ref. 30]. Let the sequence $\{r_k\}$ converge to r^*. The order of convergence of $\{r_k\}$ is defined as the supremum of the non-negative numbers p satisfying

$$0 \leq \lim_{k \to \infty} \frac{| r_{k+1} - r_k^* |}{| r_k - r^* |^p} < \infty . \qquad (5)$$

Larger values of p in (5) imply faster convergence since the distance from the limit r^* is reduced by the p^{th} power in a single step. For example, quadratic convergence (p = 2) doubles precision at each step while for linear convergence (p = 1) the reduction ratio remains constant.

Finally, if a sequence has order p convergence, then as k becomes very large, we have

$$| r_{k+1} - r^* | = \beta | r_k - r^* |^p ,$$

where β is a constant termed the convergence ratio [Ref. 30].

2. Newton's Method for Unconstrained Optimization

NLP can be solved using the well-known Newton method [e.g., Ref. 30], where the function of f being minimized is approximated locally by a quadratic function and this approximating function is then minimized exactly. Thus near the point $\bar{x}_0$, we can approximate f by the truncated Taylor series:

$$f(\bar{x}) \approx f(\bar{x}_0) + \bar{g}(\bar{x}_0)^T(\bar{x} - \bar{x}_0) + 1/2(\bar{x} - \bar{x}_0)^T H(\bar{x}_0)(\bar{x} - \bar{x}_0) .$$

Minimizing the approximation by setting the derivative equal to zero yields:

$$\bar{g}'(\bar{x}) = \bar{g}(\bar{x}_0) + \bar{g}'(\bar{x}_0)(\bar{x} - \bar{x}_0) = 0 \ ,$$

where $\bar{g}(\bar{x}_0)$ is defined as the gradient vector of the function f evaluated at the point $\bar{x}_0$ and $\bar{g}'(\bar{x}_0)$ is equivalent to $H(\bar{x}_0)$, the hessian matrix of f evaluated at x_0.

Making the appropriate symbol replacements and defining the vector $(\bar{x} - \bar{x}_0)$ as $\bar{p}$ gives us:

$$H(\bar{x}_0)\ \bar{p} = -\bar{g}(\bar{x}_0) \ . \qquad (6)$$

Equation (6) is the classical representation of the Newton method in an unconstrained setting with $\bar{p}$ often referred to as the Newton direction or the Newton step. This suggests the following iterative procedure given an initial guess $\bar{x}_0$:

Step 0. $k = 0$.

REPEAT

Step 1. Solve $H(\bar{x}_k)\bar{p}_k = -\bar{g}(\bar{x}_k)$ for $\bar{p}_k$.

Step 2. $\bar{x}_{k+1} \leftarrow \bar{x}_k + \bar{p}_k$.

UNTIL convergence.

An equally familiar result is that under the regularity assumptions given earlier this method, although well defined near a solution point, may not converge when far from a solution. One reason for such behavior is that the quadratic function is a poor approximation of f at the point $\bar{x}_k$. Since in its pure form the Newton direction contains both direction and steplength (the magnitude of the direction vector),

information based on the assumption of quadratic properties for f, either or both of these quantities may be inappropriate for the function under consideration. Traditionally, the method has been modified to enhance its convergence properties and accommodate arbitrary functions. The most frequent modification is the introduction of a search parameter λ as follows:

$$\bar{x}_{k+1} = \bar{x}_k + \lambda_k \bar{p}_k \ , \qquad (7)$$

where λ_k is chosen to minimize f in the direction $\bar{p}_k$. For points where the Newton quadratic approximation is "good," λ_k should be approximately one. When $\lambda_k \neq 1$, the ray search (7) is used as insurance that descent with respect to f is realized on the Newton direction.

Finally, Newton's method can fail if $H(\bar{x}_k)$ becomes non-positive definite at points far from a solution. In such cases the method may not yield a direction of descent and can in fact seek a relative maximum point. The method's characteristics are summarized in Figure 4.

3. Truncated-Newton Methods for Unconstrained Optimization

Since the benefits of the Newton direction are greatest in the immediate vicinity of a solution, where the quadratic approximating function best describes the actual function, an accurate determination of the Newton Direction appears unnecessary when far from a solution. The use of iterative methods to solve the Newton equations suggests a direct trade-off between computational effort and Newton solution accuracy. It

ADVANTAGES

1. Locally and quadratically convergent.

DISADVANTAGES

1. The method is not globally convergent.
2. It is not defined at points $\bar{x}$, where $H(\bar{x})$ is singular or essentially singular in a numerical sense.
3. For nonconvex problems, it does not necessarily generate a sequence of descent directions.
4. An n-dimensional linear system of equations must be solved at each iteration.

Fig. 4. Characteristics of the Newton Method.

is precisely this relationship that the Truncated-Newton class of methods [Ref. 27] seeks to exploit. At the outset of the optimization, easily obtainable but relatively inaccurate approximations to the Newton Direction are tolerated with increasing accuracy demanded as $\bar{x}_k$ approaches $\bar{x}^*$.

In order to control such a method, a measure of required accuracy is needed that reflects how "far" the present value of the objective

function is from a solution. The currently preferred measure [Ref. 27] is the relative residual

$|| \bar{r}_k || / || \bar{g}_k ||$ where $\bar{r}_k$ is defined as

$$\bar{r}_k = H(\bar{x}_k)\bar{p}_k + \bar{g}(\bar{x}_k) \ .$$

The iterative method applied to the Newton equations is therefore truncated when the relative residual is "small enough."

The basic outline of a Truncated-Newton method [Ref. 27] is given in Figure 5. Equations (8) and (9) are conditions guaranteeing "sufficient descent" [Ref. 27]. It can be shown [Ref. 31] that if these conditions are satisfied for λ_k and if $\{\bar{x}_k\}$ converges to an optimal point $\bar{x}^*$ at which $H(\bar{x}^*)$ is positive definite, then there is an iteration index $k \geq 0$ such that $\lambda_k = 1$ admissible for $k \geq k_0$ and $\{\bar{x}_k\}$ converges superlinearly (order > 1) to $\bar{x}^*$. These conditions therefore operate in conjunction with the direction finding mechanism to produce desirable terminal convergence properties (when the method does in fact converge).

Using a conjugate-gradient (CG) algorithm to iteratively calculate the search direction $\bar{p}_k$, Dembo and Steihaug [Ref. 27] propose the truncated conjugate-gradient algorithm (TNCG) illustrated in Figure 6 to serve as the minor iteration.

There are three different ways in which the TNCG minor iteration can terminate:

Given an initial solution $\bar{x}_0$:

$k \leftarrow 0$

REPEAT

Major Iteration

Compute $f(\bar{x}_k)$, $\bar{g}(\bar{x}_k)$, $H(\bar{x}_k)$

Test for convergence

IF (NOT convergence) THEN

Minor Iteration

Calculate $\bar{p}_k$ such that

$$H(\bar{x}_k)\bar{p}_k = -\bar{g}(\bar{x}_k)$$

Using a Truncated Newton iterative method

Calculate some λ_k that satisfies

$$f(\bar{x}_k + \lambda_k\bar{p}_k) \leq f(\bar{x}_k) + \alpha\lambda_k\bar{g}(\bar{x}_k)^T\bar{p}_k \quad ; \alpha \in (0,1/2) \quad (8)$$

$$\bar{g}(\bar{x}_k + \lambda_k\bar{p}_k)^T\bar{p}_k \geq \beta\bar{g}(\bar{x}_k)^T\bar{p}_k \quad ; \beta \in (\alpha, 1) \quad (9)$$

$$\bar{x}_{k+1} \leftarrow \bar{x}_k + \lambda_k\bar{p}_k$$

$k \leftarrow k+1$

ENDIF

Until convergence

Fig. 5. Truncated-Newton Method

Denoting k as the major iteration counter and $f(\bar{x}_k)$, $\bar{g}(\bar{x}_k)$, $H(\bar{x}_k)$ as f_k, $\bar{g}_k$ and H_k respectively

Minor Iteration

Step 1: $\bar{p}_0 \leftarrow \bar{0}$

$\bar{r}_0 \leftarrow -\bar{g}_k$

$\bar{d}_0 \leftarrow \bar{r}_0$

$\bar{d}^T\bar{d}_0 \leftarrow \bar{r}_0^T\bar{r}_0$

REPEAT

Step 2: $\bar{q}_i \leftarrow H_k\bar{d}_i$

IF $\bar{d}_i^T\bar{q}_i \leq \varepsilon\,(\bar{d}_i^T\bar{d}_i)$ THEN

EXIT: $\bar{p}_k \leftarrow \begin{cases} \bar{d}_0 & \text{if } i = 0 \\ \bar{p}_i & \text{otherwise} \end{cases}$

ENDIF

Step 3: $\alpha_i \leftarrow \bar{r}_i^T\bar{r}_i \,/\, \bar{d}_i^T\bar{q}_i$

$\bar{p}_{i+1} \leftarrow \bar{p}_i + \alpha_i\bar{d}_i$

$\bar{r}_{i+1} \leftarrow \bar{r}_i - \alpha_i\bar{q}_i$

IF $\| \bar{r}_{i+1} \| \,/\, \| \bar{g}_k \| \leq \eta_k$ THEN

EXIT: $\bar{p}_k \leftarrow \bar{p}_{i+1}$

ENDIF

Step 4: $\beta_i \leftarrow \bar{r}_{i+1}^T\bar{r}_{i+1} \,/\, \bar{r}_i^T\bar{r}_i$

$\bar{d}_{i+1} \leftarrow \bar{r}_{i+1} + \beta_i\bar{d}_i$

$\bar{d}_{i+1}^T\bar{d}_{i+1} \leftarrow \bar{r}_{i+1}^T\bar{r}_{i+1} + \beta_i^2\bar{d}_i^T\bar{d}_i$

$i \leftarrow i+1$

UNTIL EXIT

Fig. 6. Truncated Conjugate-Gradient Algorithm

a. The gradient vector $\bar{g}_k$ points in a direction of negative curvature ($\bar{g}_k^T H_k \bar{g}_k < 0$). In this case, the minor iteration returns with $\bar{p}_k = -\bar{g}_k$, the steepest descent direction.

b. A direction of negative curvature is encountered in the CG iteration ($\bar{d}_i^T H_k \bar{d}_i < 0$) prior to satisfying the Truncated-Newton termination criterion. The CG procedure is terminated and the current estimate $\bar{p}_k$ is used. Dembo and Steihaug [Ref. 27] show that $\bar{p}_k$ is a direction of descent.

c. The algorithm terminates with the Truncated-Newton criterion. Dembo and Steihaug [Ref. 27] show that this always occurs in the vicinity of a strong local minimum and thus is the determining factor in the rate of convergence.

Dembo and Steihaug [Ref. 27] define $\{n_k\}$ as the forcing sequence, case c above as the Truncated-Newton termination and directions resulting from either cases a or c as Truncated-Newton directions.

It can be shown [Ref. 27] that the TNCG algorithm is globally convergent and capable of coping with regions where the hessian is indefinite. The following theorem indicates that the order of convergence can be controlled with the proper choice of the forcing sequence.

Theorem 1. (Dembo and Steihaug [Ref. 27]) Properties of the Forcing Sequence.

Assume that the Truncated-Newton iterates $\{\bar{x}_k\}$ converge to $\bar{x}^*$. Then

a. $\{\bar{x}_k\} \to \bar{x}^*$ superlinearly (order > 1) if $\{n_k\} \to 0$.

b. $\{\bar{x}_k\} \to \bar{x}^*$ with order (1+t) if

$$n_k \leq c \, || \, \bar{g}_k \, ||^t$$

for some $c > 0$ and $0 < t \leq 1$.

c. $\{\bar{x}_k\} \to \bar{x}^*$ linearly (order 1) if n_k is uniformly less than one and bounded away from zero.

Theorem 1 indicates that by choosing

$$n_k = c \, || \, \bar{g}_k \, ||^t \qquad (k = 0,1,2,\ldots)$$

for some $c > 0$ and $0 < t \leq 1$, the TNCG algorithm will possess any prescribed order of convergence $(1 + t)$ between one and two. The result is an adaptive algorithm that solves NLP. When far far from a solution, $|| \, \bar{g}_k \, ||$ is large as is n_k and little effort is needed to satisfy the Truncated-Newton criterion. As $\{\bar{x}_k\} \to \bar{x}^*$, $\{\bar{g}_k\} \to 0$ which implies $\{n_k\} \to 0$ thereby forcing $\{\bar{r}_k\} \to 0$ and $\bar{p}_k$ to the Newton direction. The method therefore automatically incorporates an increasing amount of second-order information as the optimization process progresses just when such information is of greatest use.

4. Truncated-Newton Methods for Linearly Constrained Optimization

Truncated-Newton methods derived for unconstrained optimization can be extended to solve linearly constrained nonlinear problems of the form

$$\text{(LCNLP)} \quad \text{MINIMIZE } f(\bar{x})$$

$$\bar{x} \in R^n$$

$$\text{s.t.} \quad A\bar{x} = \bar{b}$$

$$\bar{l} \leq x \leq \bar{u},$$

where the function $f(\bar{x})$ is convex and separable (i.e., $f(\bar{x}) = \Sigma_{j=i}^{n} f_j(x_j)$ and each $f_j(x_j)$ is a convex function).

a. The "Reduced Problem"

In the manner of Murtagh and Saunders [Ref. 32], partition the columns of A as follows:

$$A = [B \; S \; N],$$

where the columns of B form a basis; the columns of S correspond to super-basic variables (nonbasic variables whose flow is allowed to vary between bounds); and the columns of N to nonbasic variables with flow fixed at either bound and not allowed to vary. Similarly, the other important vectors can be partitioned.

$$\bar{x} = [\bar{x}_B \; \bar{x}_S \; \bar{x}_N]^T \qquad \text{(primal solution)},$$

$$\bar{g}(\bar{x}) = [\bar{g}(\bar{x}_B) \; \bar{g}(\bar{x}_S) \; \bar{g}(\bar{x}_N)] \qquad \text{(gradient vector)},$$

$$\bar{p} = [\bar{p}_B \; \bar{p}_S \; \bar{p}_N] \qquad \text{(search direction vector)},$$

as can the function $f(\bar{x}) = f(\bar{x}_B, \bar{x}_S, \bar{x}_N)$.

This partition allows us to re-express the constraint set

$$A\bar{x} = \bar{b}$$

as

$$[B \; S \; N][\bar{x}_B \; \bar{x}_S \; \bar{x}_N] = \bar{b}$$

or

$$B\bar{x}_B + S\bar{x}_S + N\bar{x}_N = \bar{b}$$

solving for $\bar{x}_B$ (utilizing the fact that B represents a basis and thus B^{-1} exists),

$$\bar{x}_B = B^{-1}\bar{b} - B^{-1}S\bar{x}_B - B^{-1}N\bar{x}_N \ .$$

Now we are able to express f completely in terms of $[\bar{x}_S \ \bar{x}_N]$ and denote this function by $\hat{f}$.

$$\hat{f}(\bar{x}_S, \bar{x}_N) = f(B^{-1}\bar{b} - B^{-1}S\bar{x}_S - B^{-1}N\bar{x}_N, \bar{x}_S, \bar{x}_N) \ .$$

Calling $\hat{f}(\bar{x}_S, \bar{x}_N)$ the "reduced problem" or RNLP, we note the following:

$$\hat{\bar{g}} = \hat{f}'(\bar{x}_S, \bar{x}_N) = [\hat{\bar{g}}_S \ \hat{\bar{g}}_N]$$

and by the chain rule,

$$\hat{\bar{g}}_S = \frac{\partial \hat{f}}{\partial \bar{x}_S} = \frac{\partial f}{\partial \bar{x}_S} + \frac{\partial f}{\partial \bar{x}_B}\frac{\partial \bar{x}_B}{\partial \bar{x}_S} = \bar{g}_S - \bar{g}_B(B^{-1}S)$$

$$\hat{\bar{g}}_N = \frac{\partial \hat{f}}{\partial \bar{x}_N} = \frac{\partial f}{\partial \bar{x}_N} + \frac{\partial f}{\partial \bar{x}_B}\frac{\partial \bar{x}_B}{\partial \bar{x}_N} = \bar{g}_N - \bar{g}_B(B^{-1}N) \ .$$

Therefore the modified problem becomes

$$\text{(RNLP) MINIMIZE } \hat{f}(\bar{x}_S, \bar{x}_N)$$

$$\bar{x}_S, \bar{x}_N \in R^n$$

$$\text{s.t. } \bar{l}_S \leq \bar{x}_S \leq \bar{u}_S \qquad (10)$$

$$\bar{l}_N \leq \bar{x}_N \leq \bar{u}_N \ . \qquad (11)$$

As the bound constraints (10) and (11) can be handled implicitly, we essentially are now dealing with an unconstrained problem.

b. The Search Direction

A generic minimization algorithm for RNLP would calculate successive search directions from a feasible initial point, updating $\bar{x}$ until convergence. Since for iteration k of this sequence,

$$\Delta\bar{x}_k = \bar{p}_k = \bar{x}_{k+1} - \bar{x}_k$$

and from previous results we know that

$$\Delta\bar{x}_k = [\Delta\bar{x}_B \ \Delta\bar{x}_S \ \Delta\bar{x}_N] ,$$

$$\Delta\bar{x}_N = \bar{0} \quad \text{by definition,}$$

$$\Delta\bar{x}_B = B^{-1}\bar{b} - B^{-1}S\bar{x}_{S_{(k+1)}} - B^{-1}N\bar{x}_{N_{(k+1)}} - B^{-1}\bar{b} + B^{-1}S\bar{x}_{S_{(k)}} + B^{-1}N\bar{x}_{N_{(k)}}$$

$$= -B^{-1}S(\bar{x}_{S_{(k+1)}} - \bar{x}_{S_{(k)}}) = -B^{-1}S\Delta\bar{x}_S .$$

It follows that

$$\Delta\bar{x}_{(k)} = \begin{bmatrix} \Delta\bar{x}_{B_{(k)}} \\ \Delta\bar{x}_{S_{(k)}} \\ \Delta\bar{x}_{N_{(k)}} \end{bmatrix} = \begin{bmatrix} -B^{-1}S\Delta\bar{x}_{S_{(k)}} \\ \Delta\bar{x}_{S_{(k)}} \\ 0 \end{bmatrix} = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \Delta\bar{x}_{S_{(k)}} ,$$

and

$$\bar{p}_k = \Delta\bar{x}_k = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \bar{p}_{S_{(k)}} .$$

Given a feasible point $\bar{x}_0$, a Newton method to solve RNLP would involve successive solutions of a quadratic program to compute search directions.

$$\underset{p}{\text{MINIMIZE}} \ 1/2p^T H(\bar{x}_0)\bar{p} + \bar{g}(\bar{x}_0)^T\bar{p} \qquad (12)$$

$$\text{s.t.} \quad p_j \leq 0, \ x_j = l_j \ (j \in S)$$

$$p_j \geq 0, \ x_j = u_j \ (j \in B) .$$

Denoting the matrix $\begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix}$ as Z and altering (12) for RNLP implies

$$\text{MINIMIZE}_{\bar{p}_S} \; 1/2\bar{p}_S^T Z^T H(\bar{x}_0) Z\bar{p}_S + \bar{g}(\bar{x}_0)^T Z\bar{p}_S \qquad (13)$$

$$\text{s.t.} \quad p_j \leq 0, \; x_j = l_j \qquad (14)$$

$$p_j \geq 0, \; x_j = u_j \; . \qquad (15)$$

The feasibility constraints (14) and (15) limit admissible search directions for variables at bounds. As $\bar{p}_N$ is chosen to be zero and $\bar{p}_S$ can be handled explicitly during solution of (13), only the components of $\bar{p}_B$ are likly to cause trouble. Dembo and Klincewicz [Ref. 28] point out that the major difficulty is one of wasted computational effort. If after $\bar{p}_S$ is determined the induced $\bar{p}_B$ violates (14) or (15), a new basis must be found and a new search direction $[\bar{p}_B \; \bar{p}_S]$ calculated until all constraints are satisfied. They further show that if one operates using a basis with the greatest number of variables between bounds (free variables), the only nonbasic variables that cause concern, with respect to feasibility, are those moving away from a bound. Such a basis is termed a maximal basis [Ref. 28].

The solution to (13) when $H(\bar{x}_0)$ is positive definite (recall f and thus $\hat{f}$ are convex) is,

$$Z^T H(\bar{x}_0) Z\bar{p}_S = -Z^T \bar{g}(\bar{x}_0) \; .$$

$Z^T H(\bar{x}_0)Z$ and $Z^T \bar{g}(\bar{x}_0)$ are often referred to as the reduced hessian and and reduced gradient respectively.

We now have a system of equations in the unknown vector $\bar{p}$ that provides the solution to the direction finding problem for RNLP

and hence for its equivalent, NLP. Additionally, we can recover the complete p vector through the relationship,

$$\bar{p} = Z\bar{p}_S \ .$$

The Truncated-Newton Congugate-Gradient method can now be applied directly to the quadratic approximation of RNLP.

5. Primal Truncated-Newton Method Specialized for Networks

The material presented to this point is totally general and nothing in its development relies on the fact that the problem in question is a network program. With this in mind, it is now time to examine some of the special structure afforded by the network representation.

Define A of LCNLP (and RNLP) to be a node-arc incidence matrix of a directed network. The result is the usual formulation of the capacitated minimum cost network problem with a convex separable objective function. We shall label these new problems NLN and RNLN.

Any solution technique for NLN can naturally take advantage of the special structure of network optimization problems, dispense with explicit representations of the basis inverse and perform update and solution operations directly on a network specialization of the basis representation.

First consider the form of the reduced gradient of RNLN,

$$\begin{aligned} Z^T g &= [-S^T B^{-T} \ I \ 0] \ [\bar{g}_B \ \bar{g}_S \ \bar{g}_N]^T \\ &= -S^T B^{-T} \bar{g}_B + \bar{g}_S \ . \end{aligned}$$

Recognizing $B^{-T}\bar{g}_B$ as the estimates of the dual variables (denote as $\bar{w}$) and the solution to

$$B^T \bar{w} = -\bar{g}_B \ ,$$

suggests that $\bar{w}$ can be determined by solving a triangular system of equations since network bases B (and hence their transpose) can always be placed in triangular form by simple permutation. Calculation of the dual variables in this manner would allow easy computation of the reduced gradient. This is a relatively standard approach to NLN.

The solution technique just developed serves as a framework in which to embed an adaptive direction finding mechanism and thus produce a Truncated-Newton method. With the addition of control logic and procedures to monitor and manipulate the variable partition, we have all the necessary ingredients of a complete algorithm to solve NLN [Ref. 19].

Consider NLN and the associated problem RNLN. Given an initial feasible solution $\bar{x}_0$,

STEP 0: Partition $\bar{x}_0$ into basic, superbasic and nonbasic sets in such a manner that a maximal basis is established. Likewise, partition $\bar{g}$ and H.

$k \leftarrow 0$

STEP 1: Calculate the dual variable estimates ($\bar{w}$) by solving

$$B^T\bar{w} = -\bar{g}_B \quad (\bar{w} = -B^{-T}\bar{g}_B).$$

STEP 2: Compute the reduced gradient,

$$\frac{\partial \hat{f}}{\partial \bar{x}_S} = Z^T\bar{g} = -S^TB^{-T}\bar{g}_B + \bar{g}_S = S^T\bar{w} + \bar{g}_S \ .$$

STEP 3: Test for optimality on subspace defined by active superbasic variables.

IF $|| Z^T\bar{g} || \leq$ tolerance THEN optimal on current subspace

FIND those nonbasic variables eligible to enter superbaisc set. Eligibility is determined by potential for feasible displacement from bound and subsequent reduction in objective function value.

Compute $\frac{\partial \hat{f}}{\partial x_N} = -N^T B^{-T}\bar{g}_B + \bar{g}_N = N^T\bar{w} + \bar{g}_N$.

IF $(\partial \hat{f}/\partial \bar{x}_N \geq 0$ and $(\bar{x}_N)_j = l_j)$ OR

$(\partial \hat{f}/\partial \bar{x}_N \leq 0$ and $(\bar{x}_N)_j - u_j)$ for all $j \in$ nonbasic set

THEN

STOP - OPTIMAL solution has been found.

ELSE

Add the nonbasic variable(s) not satisfying the above conditions to the superbasic set.

GOTO STEP 1.

ENDIF

STEP 4: Compute a feasible, improving direction (Truncated-Newton direction) by solving

$$(Z^T H Z)\bar{p}_S = -Z^T\bar{g}$$

using a Conjugate-Gradient (CG) algorithm with the termination rule

$$\frac{\| \bar{r}_i \|}{\| Z^T\bar{g}_i \|} \leq \text{force}_k$$

where

$$\text{force}_k = \min \{\text{force}_0 , \| Z^T\bar{g} \|^t\}, \ \text{force}_0 < 1, \ t \in (0,1],$$

$$\bar{r}_i = -(Z^T H Z(\bar{p}_S)_i + Z^T\bar{g}) ,$$

i = CG iteration number.

STEP 5: Ensure all components of $\bar{p}_S$ calculated in STEP 4 maintain feasibility.

Let $(\bar{p}_S)_j = 0$ if $(\bar{p}_S)_j \leq 0$ and $(\bar{x}_S)_j = l_j$,

$(\bar{p}_S)_j = 0$ if $(\bar{p}_S)_j \geq 0$ and $(\bar{x}_S)_j = u_j$.

STEP 6: Calculate a search direction for the basic variables by solving

$$B\bar{p}_B = -S\bar{p}_S .$$

STEP 7: Find a steplength λ_k giving "sufficient descent" (Goldstein-Armijo conditions).

STEP 8: Update the flow,

$$\bar{x}_{k+1} \leftarrow \bar{x}_k + \lambda_k\bar{p}_k$$

IF $(\bar{x}_B)_j$ at bound then pivot and replace with free arc from $\bar{x}_S$ (if possible) to maintain maximal basis.

IF $(\bar{x}_S)_j$ at bound then remove from superbasic set.

$k \leftarrow k + 1$.

GOTO STEP 1.

Changes of bases are performed using the pivot mechanism of GNET [Ref. 3]. The single variable linesearch to determine λ is a safeguarded successive cubic approximation method [Ref. 33] modified to incorporate the Goldstein-Armijo conditions, (8) and (9). This particular linesearch is not crucial to the success of the method and any reasonable substitute could be used; although quite complex, it is reported to be very robust [Ref. 33]. The TNCG algorithm requires that both gradient and hessian information be available. Finally, the algorithm requires an initial feasible solution. This is not a limitation, but rather an advantage as the first solution can easily be provided by efficient linear network programs (i.e., GNET) thereby allowing the nonlinear code to be streamlined and overall computational effort reduced.

C. ENET

ENET solves network programming problems with "elastic" ranged constraints and bounded variables using a modified revised primal simplex method.

Elastic constraints can be violated by incurring a linear penalty as opposed to the rigid or inviolate constraints of the classical (network) programming problem. Brown and Graves [Ref. 20] point out that such an elastic model is a realistic and powerful portrayal of many real world situations. Hence it may be advantageous to relax some constraints (and incur a penalty) in order to satisfy others, or improve the value of the objective function. This class of models, therefore, offers the analyst complete flexibility in the formulation of NPP.

1. Ranged and Bounded Model

The traditional bounded linear (network) model (LNP)[4] is

$$\text{(LNP)} \quad \min \quad \bar{c}^T\bar{x}$$
$$\text{s.t.} \quad A\bar{x} = \bar{b} \qquad (16)$$
$$\bar{l} \leq \bar{x} \leq \bar{u}$$

Addition of upper and lower ranges on the (flow conservation) constraints (16), yield the following ranged and bounded model (RLNP):

$$\text{(RLNP)} \quad \min \quad \bar{c}^T\bar{x}$$
$$\underline{\bar{b}} \leq A\bar{x} \leq \overline{\bar{b}}$$
$$\bar{l} \leq \bar{x} \leq \bar{u} \ .$$

By introducing additional (structural) variables $\bar{y}$ and (logical) variables $\bar{s}$ as follows:

$$\bar{y} = \bar{x} - \bar{l} \ ,$$

and

$$\bar{0} \leq \bar{s} \leq \overline{\bar{b}} - \underline{\bar{b}} \ ,$$

RLNP is transformed into the equivalent equality-constrained model with translated ranges and bounds:

$$\min \quad \bar{c}^T\bar{y} + \bar{c}^T\bar{l}$$
$$\text{s.t.} \quad A\bar{y} + \bar{s} = \overline{\bar{b}} - A\bar{l}$$
$$0 \leq \bar{y} \leq \bar{u} - \bar{l}$$
$$0 \leq \bar{s} \leq \overline{\bar{b}} - \underline{\bar{b}} \ .$$

$\bar{s}$ is a vector of nonnegative slack variables one for each constraint, that measure deviation of the current constraint value from the appropriately translated upper range.

---

[4] Section III, Subsection A.

2. Elastic Model

Now consider the enhancement to RLNP where the constraint (flow) ranges are allowed to be violated. For the purposes of this paper, the penalty functions will be restricted to the linear case to maintain piecewise linearity of the objective function. Let $\bar{z}$ and $\underline{z}$ be vectors of the penalty coefficients (the i^{th} elements define the cost per unit violation for the i^{th} constraint) for the upper and lower constraint ranges respectively. The resulting model is:

$$
\begin{aligned}
\text{(ERLNP)} \quad \min \quad & \bar{c}^T\bar{y} + \bar{c}^T\bar{l} + \bar{z}^T\bar{r} + \underline{z}^T\bar{a} \\
\text{s.t.} \quad & A\bar{y} - \bar{r} + \bar{a} + \bar{s} = \bar{b} - A\bar{l} \\
& \bar{0} \leq \bar{y} \leq \bar{u} - \bar{l} \\
& \bar{0} \leq \bar{s} \leq \bar{b} - \underline{b} \qquad (17) \\
& \bar{a}, \bar{r} \geq 0
\end{aligned}
$$

The vectors $\bar{a}$ and $\bar{r}$ are (logical) artificial and surplus variables introduced to maintain equality constraints with all nonnegative variables as is the custom for the simplex method.

A mathematically equivalent model for ERLNP is

$$\min \quad \bar{c}^T y + \bar{c}^T\bar{l} + \bar{z}(A\bar{y} - \bar{b} + A\bar{l}) \qquad (18)$$

with

$$
\bar{z} = \begin{cases} \bar{z} \text{ if } A\bar{y} \geq \bar{b} - A\bar{l} \\ -\underline{z} \text{ if } A\bar{y} < \underline{b} - A\bar{l} \\ 0 \text{ otherwise .} \end{cases}
$$

$\bar{z}, \underline{z} > 0$ (for convexity) .

This is known as the Lagrangian form, and illuminates the fact that $\bar{z}$ and $\underline{z}$ are actually bounds on the variables of the dual formulation.

A graphical representation of the elastic cost implication for each constraint of ERLNP is given by Figure 7.

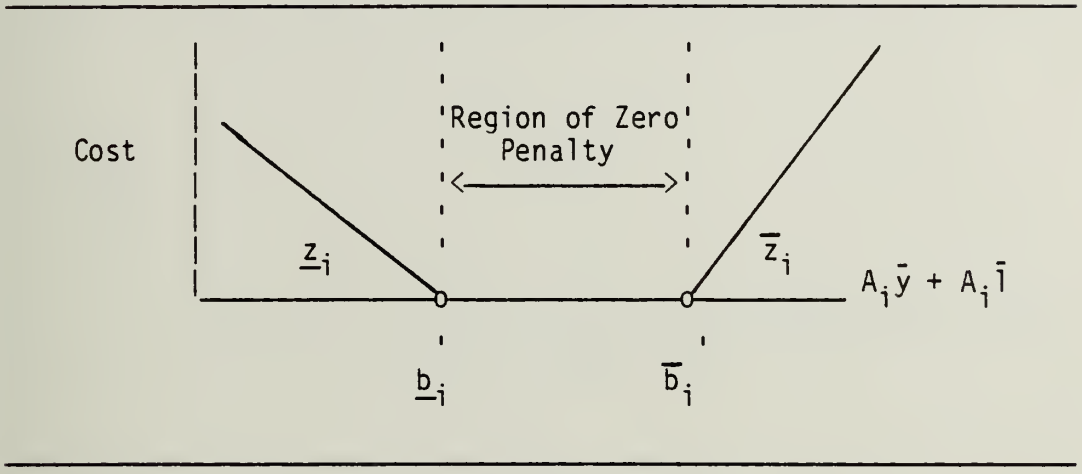


Fig. 7. An Elastic Ranged Constraint

In this sense, with $\underline{z} = \overline{z} = \infty$, each constraint can be depicted as in Figure 8 where the discontinuities at the upper and lower ranges indicate that the ranges are rigid as in (RLNP) and cannot be violated (i.e., an infinite penalty beyond the defined range). Flow within the allowable range incurs zero penalty.

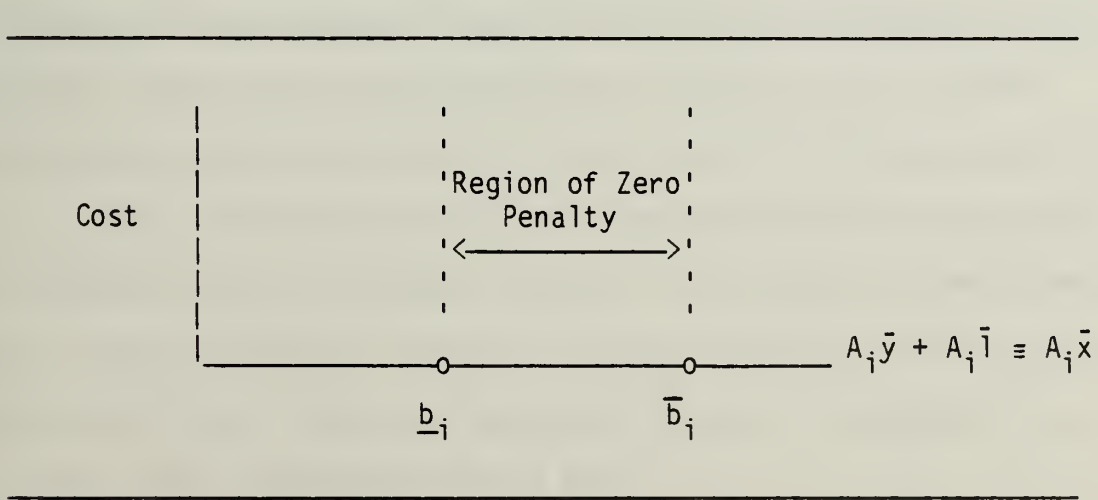


Fig. 8. A Rigid Ranged Constraint

When $\overline{b}_i = \underline{b}_i = b_i$, the range collapses to a point as shown in Figure 9 and the model specializes to LNP.

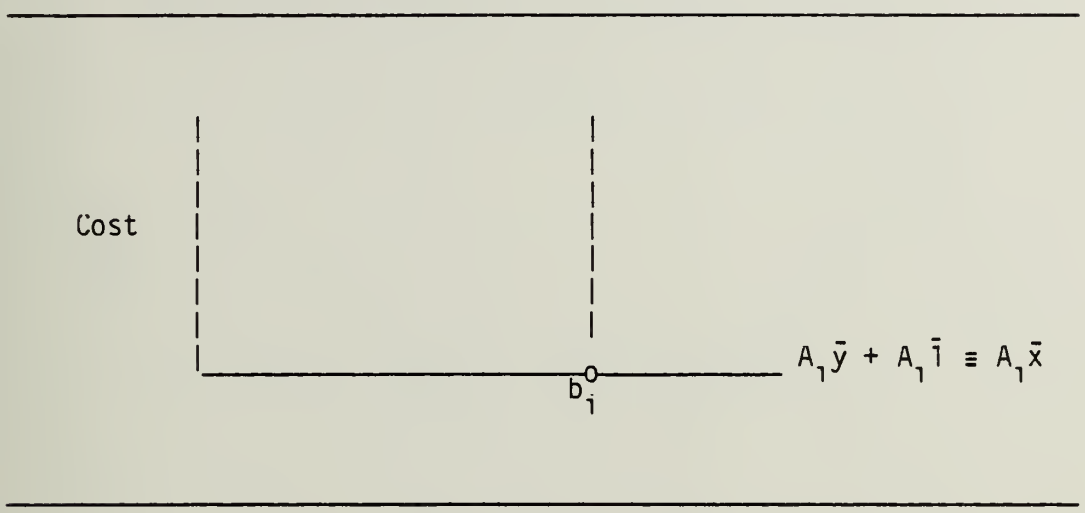


Fig. 9. Classical Equality Constraint

Consider the relationship between the logical variables $\bar{s}$, $\bar{a}$, and $\bar{r}$ in ERLNP. The slack variable s_i measures the distance a solution is from the upper range of constraint i. As given by (17), s_i has an upper bound equal to the difference between the upper and lower ranges. The artificial variable a_i measures the distance a particular solution is below the lower range while the surplus variable r_1 measures distance above the upper range (both are unbounded variables in the current presentation). Figure 10 shows this relationship for a given constraint. s_i, a_i, and r_i are mutually exclusive in any basic solution since they are evidently linearly dependent columns. Additionally, a_i and r_i must be equal to zero if non-basic, while s_i can be non-basic at either zero or its upper bound $\overline{b}_i - \underline{b}_i$. Thus, for any given solution, the status of s_i, a_i, and r_i may be summarized (Figure 10):

1. Upper range violated by r_i at cost $\overline{z}_i$ r_i ($a_i = s_i = 0$).

2. No violation ($a_i = r_i = 0$).

3. Lower range violated by a_i at cost $\underline{z}_i$ a_i ($s_i = \overline{b}_i - \underline{b}_i$, $r_i = 0$).

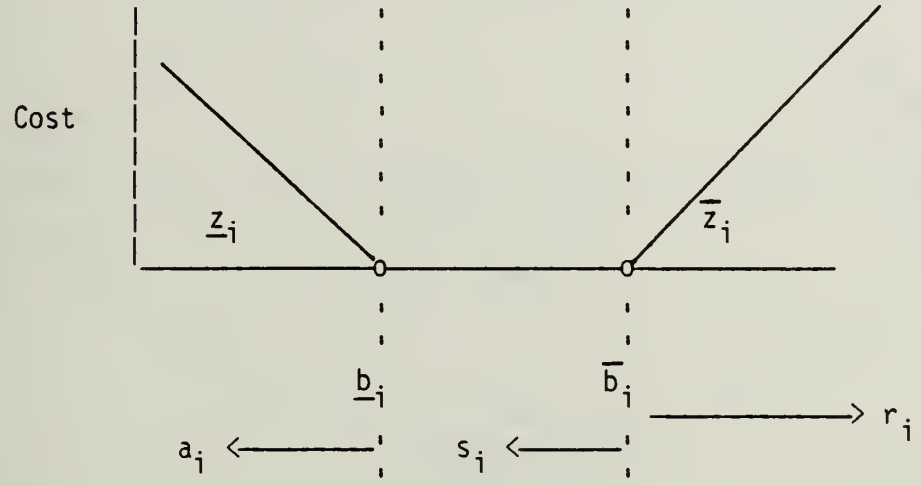


Fig. 10. Relationship Between Logical Variables for ERLNP.

3. Elastic Primal Revised Simplex Specialized for Networks

All solutions to ERLNP which satisfy bounds ($l_j \leq x_j \leq u_j$) are feasible since the artificial and surplus variables are unbounded. Therefore, a solution technique for ERLNP does not need to be partitioned into the usual two phases: (I) where feasibility is achieved, and (II) where optimality is obtained while maintaining feasibility. Any starting solution satisfying bounds $\overline{l}$ and $\overline{u}$ will serve to begin the solution process. Essentially, ENET employs the primal revised simplex technique of GNET with modifications in logic and data structure to accommodate the additional arcs (variables) implied by the elastic model. Figure 11 shows the logical arcs employed for each node. For any solution, all

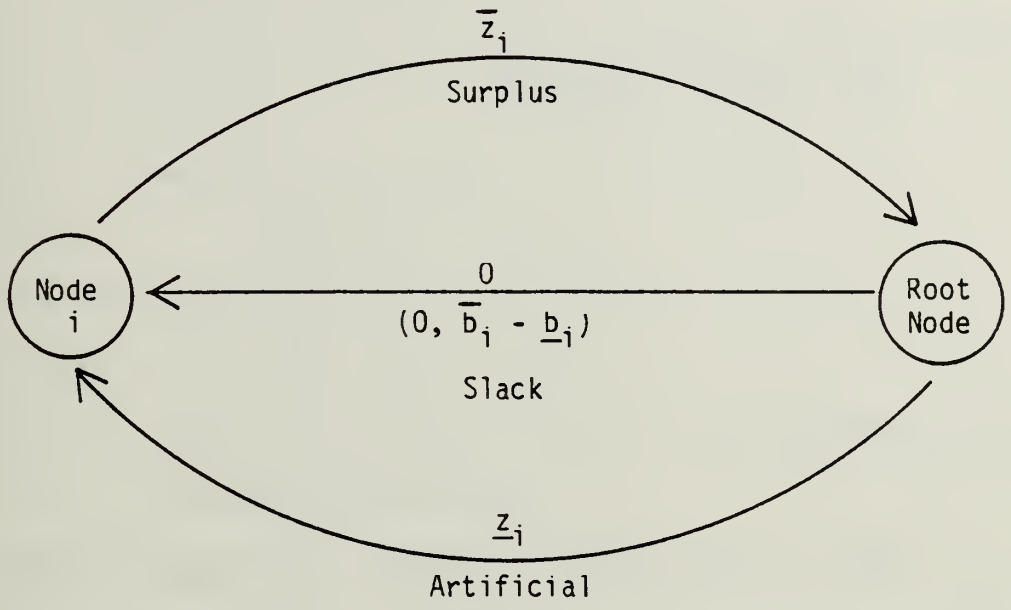


Fig. 11. Logical Arcs Generated for Node i

required information for each (flow conservation) constraint and its penalty contribution to the objective function is generated logically from:

1. which logical arc (if any) is in the basis, and
2. which bound the slack arc s_i assumes when non-basic.

Brown and Graves [Ref. 20] describe the following Elastic Primal Simplex Network Algorithm:

STEP 0: Select a starting solution.

REPEAT

STEP 1: Price Out. Given the dual variables $\bar{w}$, the reduced cost of each explicit arc is:

$$r_k \leftarrow c_k - w_i + w_j$$ (arc k orientated i to j, or non-basic arc k at upper bound orientated j to i)

and for logical arcs,

$$r_i \leftarrow \begin{cases} \underline{z}_i + w_i & \text{for } a_i \\ 0 + w_i & \text{for } s_i \\ \bar{z}_i - w_i & \text{for } r_i \end{cases}.$$

Select an incoming arc with $r_i < 0$ or terminate with the current solution OPTIMAL.

STEP 2: Ratio Test. Determine outgoing arc as in GNET.

STEP 3: Update. Same as GNET, substituting obvious specializations for updates involving logical arcs.

UNTIL OPTIMAL

The only data required by this enhanced elastic model in addition to classical GNET are for each node the new penalties, the difference between the upper and lower ranges, and whether s_i is reflected.

The resulting algorithm and data structures are well-suited for microcomputer implementation. The moderate increase in data storage is compensated by a significant model enrichment. In particular,

1. Total supplies and total demands no longer need be equal, nor must supply nodes be connected to demand nodes by paths of sufficient capacity to insure classical feasibility. Infeasible problems are intrinsically diagnosed and optimally treated.

2. All problems possess feasible primal solutions in the extended elastic formulations, and this provides reliable bounds for dual solutions.
3. All bounded problems possess optimal solutions yielding optimal dual solutions.
4. Informal relaxation (e.g., Lagrangian Relaxation) methods are naturally accommodated in this context.
5. Formal decomposition methods are provided much more robust primal and/or dual proposals.
6. (Mixed) integer models produce strictly feasible solutions in this extended Lagrangian context.

and perhaps most important:

7. Solutions are reliable, inexpensive, and managerially appealing.

Finally, as we shall see, the complete Lagrangian objective function yields a unifying perspective of solution properties and intrinsically gives profound information to the analyst or algorithms using ENET.

D. UNET

UNET solves elastic ranged bounded linear network problems where certain arcs are designated as "l-u" or binary arcs. These arcs can admit flow at one of two possible values--the lower or upper bound, hence the label "l-u". Consider first the simplest bounded "l-u" model:

$$\text{(LULNP)} \quad \min \quad \bar{c}^T \bar{x}$$

$$\text{s.t.} \quad A\bar{x} = \bar{b}$$

$$\bar{l} \leq \bar{x} \leq \bar{u}$$

$$\bar{x} \equiv \{\bar{x}_f, \bar{x}_u\}, \ \bar{x}_u \in \{\bar{l}, \bar{u}\} \quad \text{(integer restriction)}.$$

By adding the now familiar elastic range framework LULNP becomes

$$
\begin{aligned}
\text{(ELULNP)} \quad \min \quad & \bar{c}^T\bar{y} + \bar{c}^T\bar{l} + \bar{\bar{z}}^T\bar{r} + \bar{\underline{z}}^T\bar{a} \\
\text{s.t.} \quad & A\bar{u} - \bar{r} + \bar{a} + \bar{s} = \bar{b} - A\bar{l} \\
& \bar{0} \leq \bar{y} \leq \bar{u} - \bar{l} \\
& \bar{0} \leq \bar{s} \leq \bar{b} - \bar{\underline{b}} \\
& \bar{a}, \bar{r} \geq \bar{0} \\
& \bar{y} \equiv \{\bar{y}_f, \bar{y}_u\}, \ \bar{y} \in \{\bar{l}, \bar{u}\} \quad \text{(integer restriction).}
\end{aligned}
$$

Such a formulation is termed a mixed integer problem (MIP) since there are both fractional (with respect to flow bounds, $\bar{x}_f$) and integral (again with respect to flow bounds, $\bar{x}_u$) arcs present. Note that the cost coefficients $\bar{c}_u$ associated with $\bar{y}_u$ may be interpreted as fixed charges in the sense that for admissible solutions, the cost contribution is either $\bar{c}_u^T\bar{l}$ or $\bar{c}_u^T\bar{u}$ for each "l-u" arc, and that each "l-u" arc is essentially equivalent to a binary decision variable.

Brown and Graves [Ref. 20] employ an enumeration technique to solve ELUNLP which exploits the elastic model structure and produces excellent solutions satisfying the integer restrictions with very little computational effort.

Their approach is analogous to classical branch and bound [e.g., Ref. 1] with the following specializations and supporting observations:

1. Any continuous relaxation of ELULNP (with integrality violated by one or more arcs in $\bar{y}_u$) provides a lower bound for the value of an optimal solution to ELULNP.
2. Any continuous relaxation of ELULNP can be rounded to an integer solution (satisfying integrality for $\bar{y}_u$) with very little

effort. Further, such rounded solutions are all admissable (feasible in the extended elastic sense).

3. Restrictions of ELUNLP (with one or more arcs in $\bar{y}_u$ fixed at a bound) are all admissible (elastically feasible) and possess solution values no higher than the lower bound of their respective relaxations.

Thus, enumeration by branch and bound may be organized:

STEP 0: Relax (free) all integrality restrictions on $\bar{y}_u$, prepare for storage of an incumbent solution, set the lower bound for solution value to $+\infty$.

REPEAT

STEP 1: Solve ELUNLP with current restrictions.

STEP 2: Improve lower bound on solution value, if possible.

STEP 3: Round solution (heuristically) to satisfy integer restrictions.

STEP 4: Compare rounded solution with the incumbent, replace incumbent if possible.

STEP 5: IF current solution value is worse than current lower bound, go to STEP 7.

STEP 6: Fix a variable. Heuristically select a free member of $\bar{y}_u$ and fix it at a bound. Go to STEP 1.

STEP 7: Heuristically select STEP 8 or STEP 9.

STEP 8: Reverse a variable. Select a previously fixed variable and reverse it to its opposite bound, go to STEP 1. If no fixed variable exists, go to STEP 9.

STEP 9: Backtrack. Select a reversed variable and free it. Go to STEP 1.

UNTIL TERMINATION.

Note that the heuristic decision rules involve the method of rounding a solution and the selection criteria for candidate variables to fix, reverse, and free (Figure 12). Within this framework, particular heuristics yield a wide variety of enumeration strategies. The particular strategy chosen for microcomputer use:

1. rounds the current restricted solution in three passes, each of which selects variables from a class defined in terms of θ, where $0 \leq \theta \leq .5$ and $\theta u \leq \bar{y}_u \leq \bar{u}$ or $\bar{l} \leq \bar{y}_u \leq \theta\bar{l}$.

   Class 1: nearly integral ($0 \leq \theta \leq .2$)

   Class 2: fractional ($.2 < \theta \leq .4$)

   Class 3: ambivalent ($.4 < \theta \leq .5$)

   The rounding heuristic sequentially exhausts variables from each class and rounds using a "minimal regret function," rounding away from the worst penalty.

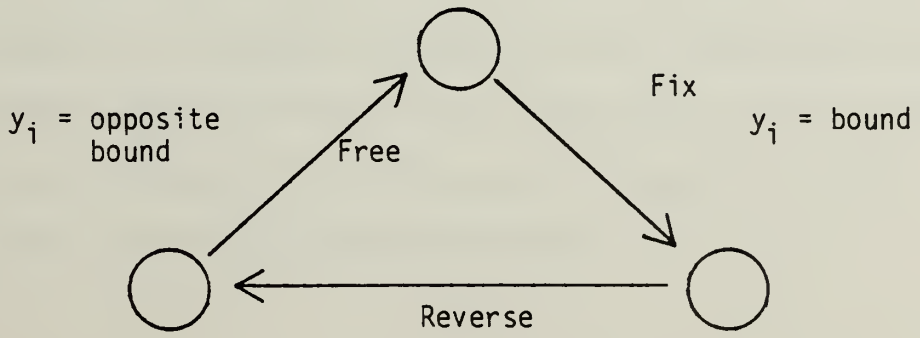


Fig. 12. Primitive Enumeration Restrictions

2. variables are selected for fixing by minimal global regret, and
3. variables are fixed, reversed and freed via a LIFO List operating exclusively on the last entry in the list:

    Fix: Push fixed variable on LIFO.

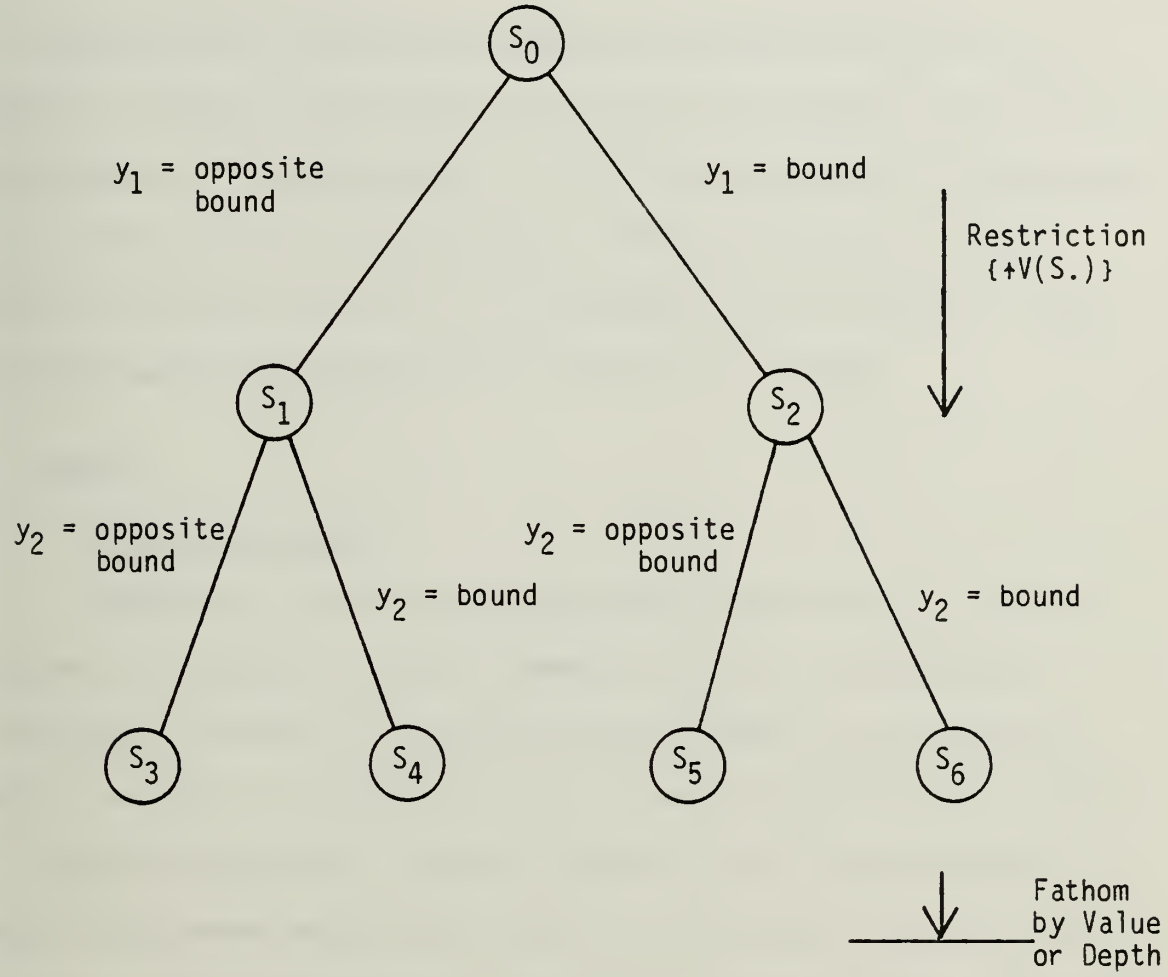
    Reverse: Update status of top variable on LIFO.

    Free: Pop reversed variable from LIFO, mark it as freed (Figure 12).

Also note that there are only depth and value-motivated fathoming rules (decision rules for reversal or backtracking); feasibility plays no role in the enumeration except via the value of the Lagrangian objective function.

Finiteness of this class of enumeration methods is evident as long as the fix/reverse/backtrack mechanism cannot yield successive solutions with identical restrictions.

What is not immediately apparent is the remarkable effectiveness of these heuristics with the elastic enumeration model. Integer solutions and lower bounds of excellent quality are empirically produced quite early in the enumeration effort, permitting routine early termination of the search based on an optimality tolerance or on a maximum depth (permissible number of fixed variables in any restriction); tuning of the method is easily accomplished via these two limits and the elastic penalties used to express the underlying model.



S_0 ≡ "relaxed" solution, V(S.) solution values

Fig. 13. Generic Enumeration Tree (Fixed Order)

IV. PACKAGE DESCRIPTION

The mathematical programming package described here provides an integrated repertoire of minimum-cost network flow models for use with a microcomputer. Various interface and control modules are also described which reduce the user's workload and automate the solution process. Extensive facilities for data file management and provisions for incorporating custom problem generators complete the package.

A. OVERVIEW

1. Package Components

Actually a suite of separate programs coordinated by a master program, the package is automated wherever possible and completely interactive. Designed in a highly structured manner, each program is modular, relatively standardized and, with minor modifications, capable of independent operation. Routines common to more than one program or subject to frequent modification reside in the system library. These features simplify modification or deletion of existing modules, addition of new programs, or even the incorporation of the entire package (or some subset) into a larger structure. A macro view of the package is given in Figure 14.

THE APPLE II microcomputer version of the package, exclusive of data files, spans two disk volumes (removable floppy disks). The Pascal operating system, system library, master program, and all solution programs reside together on a volume that is always on-line. The editor and associated preprocessor programs are on a second volume that can

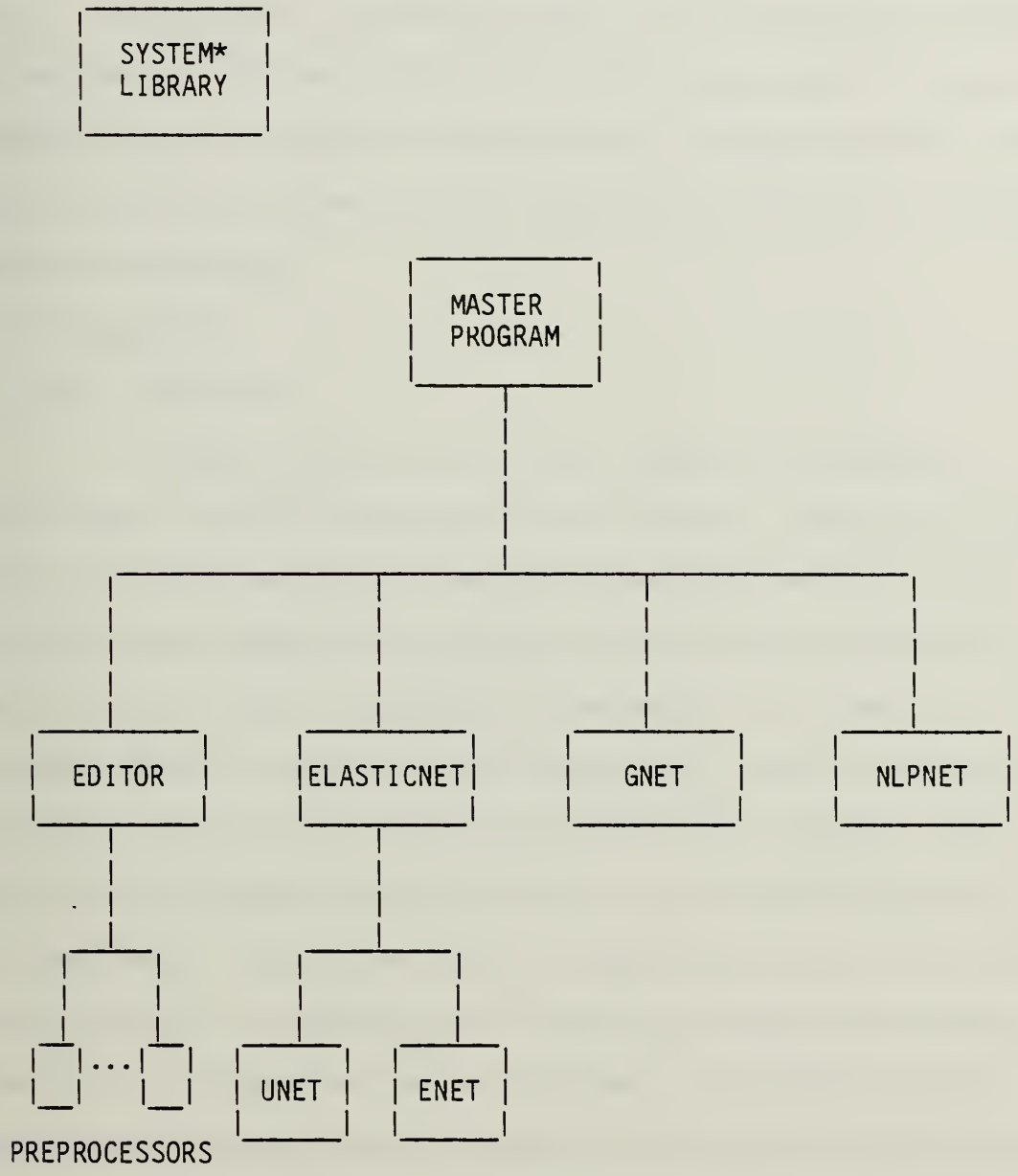


Fig. 14. Package Block Diagram

be taken off-line after the desired program is loaded into main memory. Data files occupy additional volumes present only for read/write operations. This partition is dictated by the size of the package code and has been chosen to minimize disk manipulation requirements in a two-drive system (the minimum practical configuration). Package operation is thus able to proceed with volume exchange required only for data file access and editing procedures.

2. Data Files

a. Structure

Although several quite distinct models are supported, a common format is possible through the use of Pascal's facility of variant records. A Pascal record is a compound structure of arbitrary types of data which, when composed of types with partly identical components, is termed a variant record. A portion of the record is the same for all occurences while the remainder (the variant part) may differ depending on the value of an indicator variable (also part of the record). Wirth [Ref. 34] gives an excellent description of such data structures and their employment. This allows use of a single data structure for inter-program transfer of information with specific portions of the package extracting only those items they actually need. Each program then converts this input data into the required internal data representation. Solution technique selection can thus be accomplished without user intervention as the data record contains enough information to determine problem class and a record can be accessed by any portion of the package. The use of the same type record for all problems provides streamlined

data access procedures without sacrificing the efficiency of custom internal data structures for each program.

A typical data file is a collection of randomly accessable records, each of which has three possible structures: (1) header, (2) arc, and (3) node. The Pascal interpretation of this scheme is described in Appendix A. Although only one header record is allowed per file, any number of arc and node records (subject to free disk volume space) can be contained in a single file without regard to order. All arcs of the model must be explicitly represented in the data file; however, only those nodes with attributes not equal to default values must be included. Solution programs assign appropriate default values to all nodes and then process the input file making note of the nodes that deviate from default settings. A disk volume dedicated to one file can accommodate approximately three thousand such records (APPLE II).

Constructed by the EDITOR during file creation, the header record describes the problem represented by the file. Problem name, problem type, number of nodes, number of arcs, and access history are included in this record.

Each arc record contains the arc name, source and destination nodes, bounds on flow, initial flow, and cost information. Depending on the type of arc represented, the cost data varies. For linear cost functions, only the cost per unit flow and the "l-u" status are needed. Nonlinear cost functions require specification of the function identification (assigned number in the function library) and coefficient structure.

Node records specify name, identification number in the model, type, flow requirements, flow range, and penalty for range violation.

The identification number must be unique for a given model (enforced by the EDITOR on file creation and update). Flow requirements determine the node type as follows: zero flow--transshipment, positive flow--supply, and negative flow--demand.

Internal data representations will be presented as the various programs are discussed.

b. File Naming Conventions

In order to reduce operator workload, a menu-driven data file selection technique is employed. Each data file name, regardless of the type model it represents, contains the suffix ".net." Whenever a file is to be accessed, a directory for the on-line volumes is displayed, files with this suffix are identified, and a single keystroke selects the desired file. This approach ensures that only files compatible with the package are accessed and greatly simplifies their retrieval (from the user's point of view). The EDITOR program appends the required suffix to user-designated file names upon file creation.

3. User Friendly Features

Menu-driven displays and single keystroke option selection make the package easy to use. On input, all user entries are examined for errors in the sense of range, type matching (numeric vs. alpha), and context as applicable.

Two general menu formats are standard throughout the package. The first is used for selecting courses of action. In such a situation, one of the displayed items must be selected before the program continues. The option selection menu from the EDITOR program serves as an example of this process and is shown in Figure 15. A common decision point in

```
                    EDITOR PROGRAM

OPTIONS
#######

  A(lter an existing file
  B(rowse through file
  C(reate a new file
  R(emove a file [permanently]
  T(ransfer a file

  <ESCAPE> EDITOR program and return to MASTER program

                  OPTION DESIRED

                  ---[      ]---
```

Fig. 15. Typical Option Selection Menu

optimization programs concerns the setting of program parameters. The second menu format deals with this by displaying the default values or choices as appropriate. To change a particular value, the user enters the menu-designated symbol associated with the parameter in question. If the parameter is an ON/OFF choice, the change is made when the symbol is entered; for numerical quantities, the user is prompted for the new value. Menu updating occurs automatically until the user indicates that all values are correct. Figure 16 illustrates a typical parameter control menu. Choices from either format may invoke further sub-menus. Only selections contained in the current menu are admitted: when the user enters a choice symbol not depicted by the menu in use, an appropriate error message is issued.

```
                    ELASTICNET

OPTIONS
#######

  C(omplete printout      -->    ON
  D(etailed printout      -->    OFF
  F(ile output            -->    OFF
  H(ard copy              -->    OFF
  I(nternal array dump    -->    OFF
  M(odel alteration       -->    OFF

  Are options satisfactory ?   Y(es  or  N(o   -->
```

Fig. 16. Typical Parameter Control Menu

B. CONTROL AND UTILITY PROGRAM DESCRIPTIONS

1. Master Program

The master program orchestrates package operation by passing control to appropriate programs as requested by the user (either directly or indirectly). This is accomplished with two levels of command internal to the master program as shown in Figure 17. The outer command level allows the choices of (1) data file manipulation, (2) problem solution, and (3) package exit. The first and third choices transfer control to the EDITOR program and the Pascal operating system respectively. A problem solution request activates the solution command level and package flow proceeds as depicted in Figure 18.

Problem solution begins with user selection of the input file. Once the data file has been identified, the course of action is determined by problem size (number of nodes, number of arcs) and type as indicated

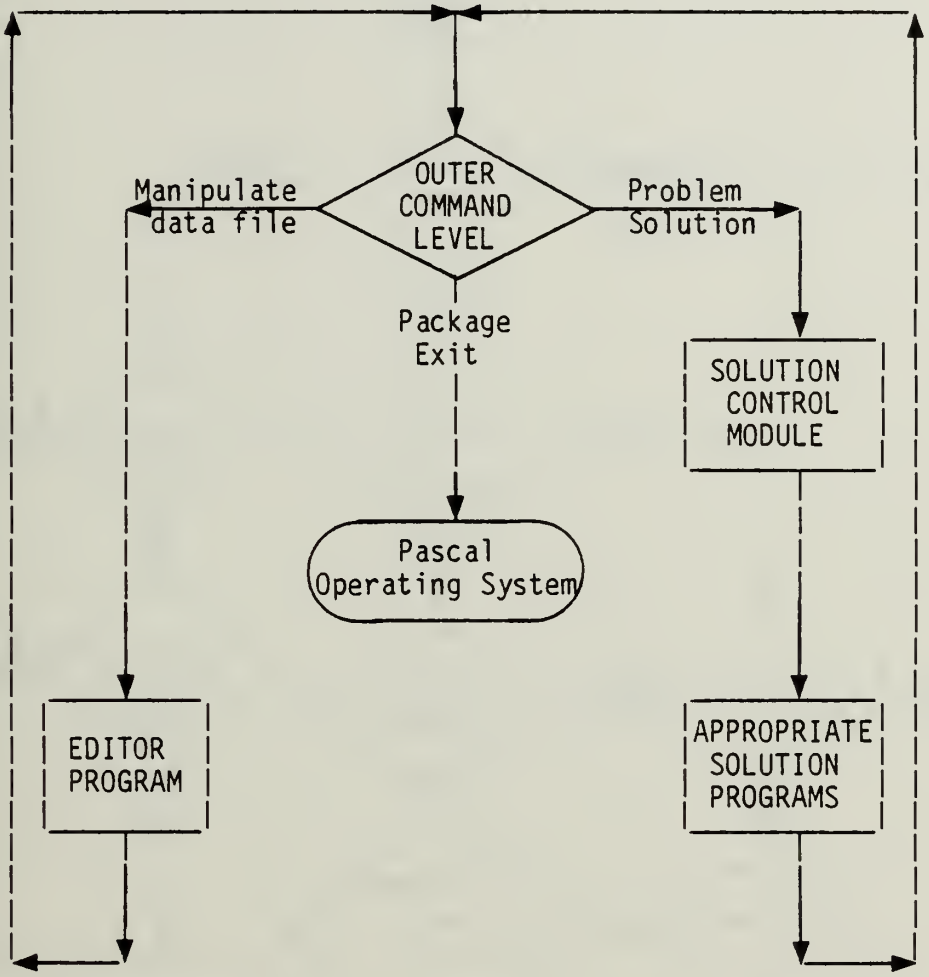
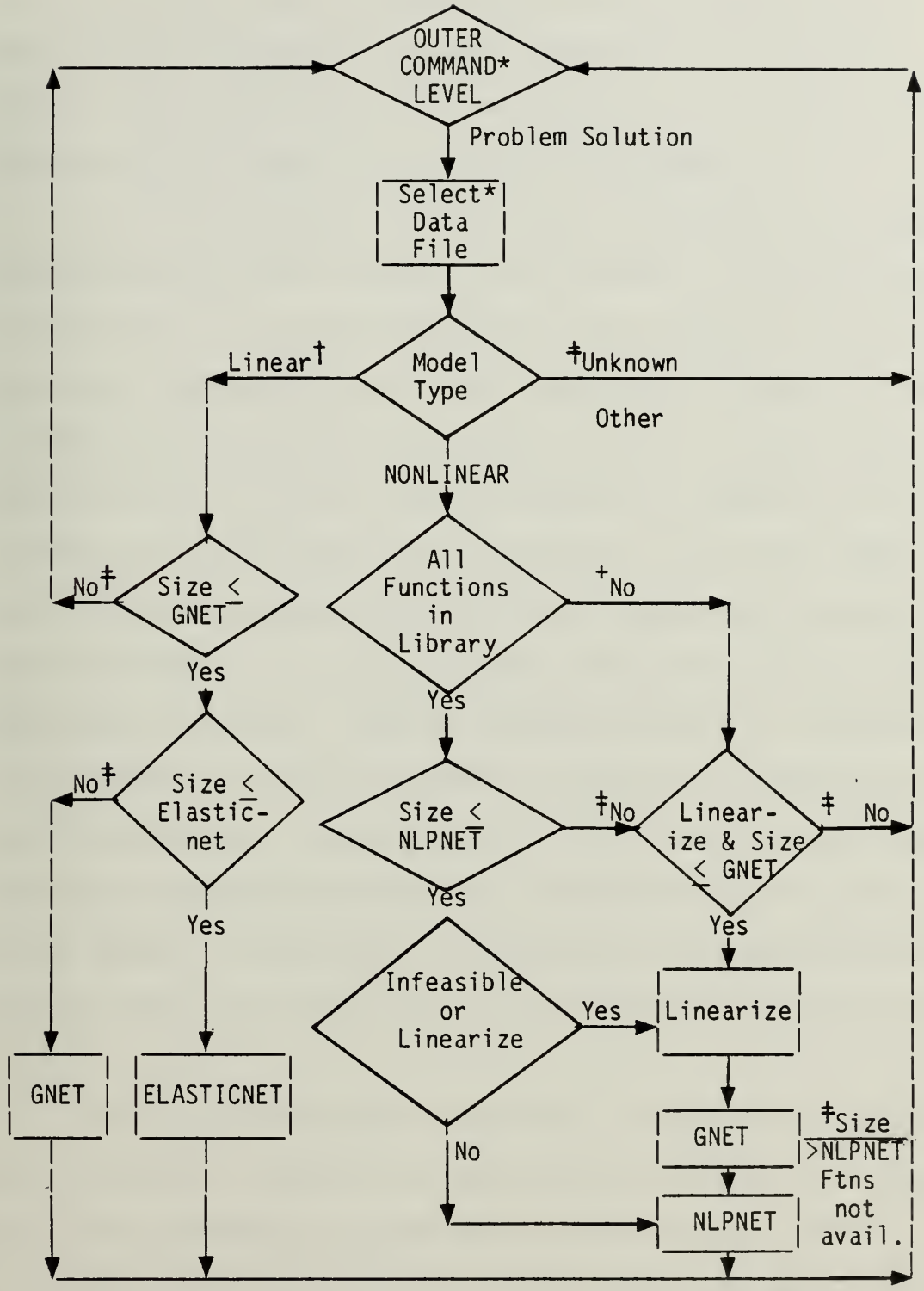


Fig. 17. Outer Command Logic



* indicates user interaction required (otherwise automatic)
† indicates elastic & "o-u"
‡ indicates warning message issued

Fig. 18. Solution Process Logic

by the file header record. Linear problem files are those with all linear arc cost functions to include problems with explicit elastic ranged or "l-u" constraints. Nonlinear elastic or "l-u" problems are not supported and are treated as ordinary nonlinear models.

All linear problems, size permitting, are routed to ELASTICNET (the combination ENET/UNET program), otherwise GNET is invoked (again size permitting). Nonlinear problems are first examined to ensure that their size is commensurate with NLPNET capabilities and that all required cost functions are resident in the system library. If either one of these tests fails, the user is given the option to linearize and use only GNET for an approximate solution (assuming size is within GNET limits). Once it has been decided that NLPNET can safely be called, initial flow feasibility is determined by a straightforward node flow conservation calculation. Feasible problems go directly to NLPNET for solution, however, infeasible problems are temporarily linearized and a feasible starting point obtained by GNET prior to being sent to NLPNET. In the event that GNET determines that no feasible solution exists, the solution process is terminated. Error messages are issued whenever the outcome of the process differs from that expected for the type file in question.

A variety of elementary linearization options are offered as noted in Figure 19. The arc cost functions are evaluated at user designated points and combined as specified to obtain point or interval approximations which then serve as cost coefficients for GNET. These costs are discarded after use by GNET. It should be noted that this is not a piecewise linearization and is not intended to be a complete

1. Point gradient eval @ LCC* of bounds (t*l + (1-t)*u).
2. Point gradient eval @ midpoint of bounds (t = 0.5).
3. Point gradient eval @ LCC of lowerbound and present flow.
4. Point gradient eval @ LCC of present flow and upperbound.
5. Interval linearization from lowerbound to specific point.
6. Interval linearization over entire bound interval.
7. Assign zero cost to all arcs.

* LCC = Linear Convex Combination

OPTION DESIRED

---[ ]---

Fig. 19. Linearization Options

solution technique. GNET can support piecewise linearizations if the model is explicitly described by addition of the appropriate arcs. The user can request that all nonlinear problems be linearized, regardless of feasibility status, by overriding a default parameter.

The program calls depicted in Figure 18 are automatically generated by the solution control module of the MASTER program. Information such as data file names and package control instructions are passed to activated programs through the Pascal operating system. Once control is passed to a program, this information is retrieved and the subject file is verified as still being on-line. In the event that the volume containing the data file has been moved off-line, the program issues an error message and awaits user action.

2. EDITOR Program

The EDITOR program allows the user to create, alter, transfer, and examine data files as indicated in Figure 20. Since this portion of the

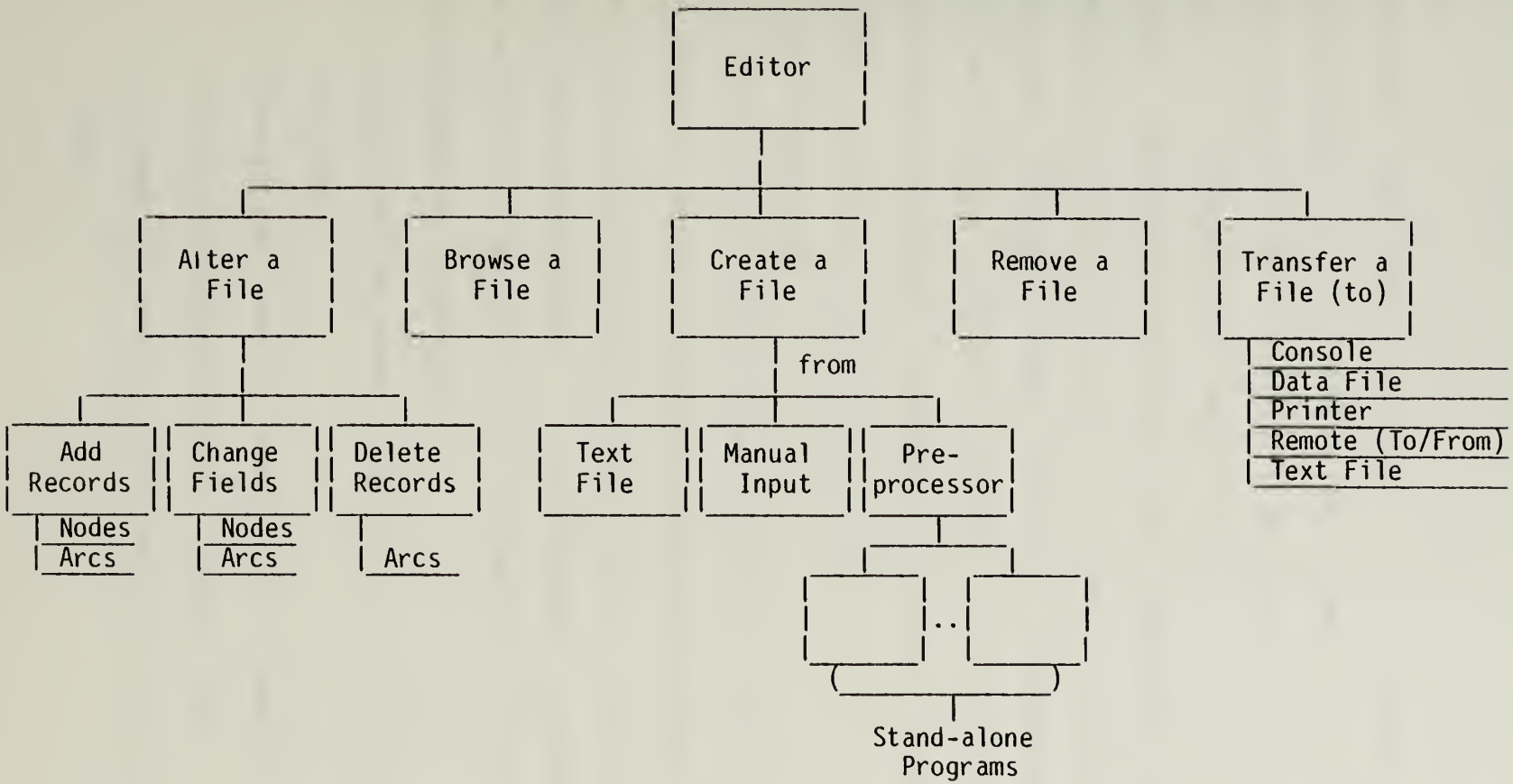


Fig. 20. Editor Block Diagram

package requires the most user participation, every attempt has been made to make the program as user-friendly as possible. In addition to employing the standard interactive techniques, the EDITOR program input format can be reconfigured to conform to the user's needs. This redefinition of the input field order can be effected at any time during the editing process. For example, one file can be created utilizing the default input order, reconfiguration performed, and a second file altered with a completely different input field order. Instead of reordering the raw data, one merely changes the order in which the program expects data fields to be received. Additionally, fields can be omitted entirely from the input format and default or user-definable constant values assigned. Input field order for arcs and nodes may differ.

a. File Creation

Files can be created from text (human readable) files, keyboard input, or through the use of preprocessor programs. For all modes of file creation, the program keeps track of the filetype by typing the arcs as they are received (i.e., one nonlinear arc changes a linear file to nonlinear). Input from text files and the keyboard is examined for the following inconsistencies:

1. Prohibited node numbers (nonpositive or greater than number of nodes.
2. Inconsistent upper and lower limits on quantities (lower limit not less than or equal to upper limit).
3. Assignment of quantity values outside of established limits.

b. File Alteration

Records can be added, deleted and their individual data fields updated. Addition of records is allowed for both node and arc records; however, only arcs may be deleted. This restriction is imposed because no connectivity analysis is performed by the EDITOR on the network representation resulting from alterations. Deletion of arcs can isolate portions of the network, but the solution programs accommodate this situation effectively. Removing nodes requires an exhaustive search of the data file to detect arcs that must also be deleted. This capability is not presently supported by the EDITOR.

c. File Transfer

Data files can be transferred to the console, printer, another data file, a text file, or a remote computer. The data field format in effect at transfer governs the order of field transmission so the package can be linked to data bases with diverse format requirements. File transfers to and from other computers are limited to text files only, thus the ability to convert between data files and text files has been provided. A stand-alone file transfer program [Ref. 35] has been used for such communication with great success.

3. Preprocessor Programs

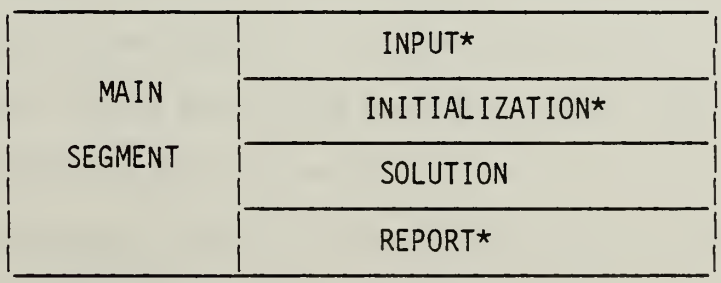
These programs are created for specific models to assist in data base reduction. For example, if data on a certain network exists in raw form such as physical measurements on the actual entities represented by the nodes and arcs of the model, the user can write a custom program to translate such information to a form suitable for package use and then execute the conversion directly from the EDITOR program. A catalog of

existing preprocessor programs is maintained by the EDITOR program and presented to the user upon request. After selecting the new data file name, control is passed to the desired preprocessor and returned to the EDITOR upon completion of preprocessing.

Facilities are provided by the EDITOR to maintain the preprocessor catalog (a data file) and manually activate uncataloged preprocessors (which are then automatically added to the catalog). The preprocessors are stand-alone programs not constrained to reside on one of the package volumes with no (package imposed) limits on their number or individual size.

C. SOLUTION PROGRAMS

Each program is partitioned into four segments as shown in Figure 21. The main segment (of the particular solution program) calls subordinate segments into memory sequentially, thereby maximizing memory available for data. The input, initialization, and report segments contain all procedures that communicate with either the user or other portions of the package. Solution segments are therefore minimal representations of their respective algorithms and data structures, and are devoid of any nonessential information.



*User-friendly structures used

Fig. 21. Generic Solution Program Segment Map

The report segment automatically updates the data file with the new solution and displays the results of the optimization on user-designated peripherals. Detailed reports of the solution process may be provided upon user request. Subsequent program calls are controlled by logic contained in the report segment.

1. GNET

This program is constructed so that both linear and nonlinear problems are processed in essentially the same manner. Two minor differences are accommodated by a flag that keys on the problem-type field of the header record. First, nonlinear files sent to GNET for solution have the linearized arc costs placed in the initial-flow field of the arc data record as GNET does not use this field for linear problem input data. Additionally, on obtaining a feasible solution, control is passed to NLPNET vice the MASTER program when GNET terminates from a nonlinear problem.

a. Input Requirements

Upon activation, GNET extracts the information listed in Figure 22 from the input data file. Certain compacting operations take place, as each arc record is processed, to reduce storage requirements. The arc flow range is stored as one number, the upper bound, after translation by the lower bound. Also, the destination node list is maintained with all arcs having the same destination node stored in contiguous locations. This allows the list to be a node-length array instead of an arc-length array. The arc costs and source nodes are read directly into arc-length arrays.

ARCS:

Source node number
Destination node number
Lowerbound on flow
Upperbound on flow
Cost information

NODES:

Net flow

Fig. 22. GNET Input Data Requirements

Node information, other than net flow, is ignored as GNET does not support ranged constraints. In the event that total supply does not equal total demand, the program will terminate abnormally. GNET operates with integer arithmetic; therefore all input values not explicitly integer are truncated upon receipt.

b. Internal Data Structure

GNET introduces an artificial node called the "root" to complete the basis tree. Three node length arrays are used to represent the basis tree and provide a mechanism to easily traverse the tree. The predecessor array defines the next node on the path from a given node to the root. For example, predecessor (i) contains the node number of the predecessor of the i^{th} node. The depth array stores, for each node, the number of nodes on the path to the root. Finally, the traversal array provides for each node the next node to evaluate during forward-substitution of the basis. This array gives a sequence of nodes which, combined with the respective node predecessors, define an upper triangulation of the basic arcs. The i^{th} element of this array is the next

node that would be visited from node i in a classical preorder[5] traversal. Arrays to store flow on basic arcs and the simplex dual variables complete the data structure. Figure 23 illustrates the basis representation and Figure 24 describes the various arrays.

These arrays are used by GNET to perform the simplex operations with elementary algebraic and logic operations. A detailed description of this data structure and its employment is given by Bradley, Brown, and Graves [Ref. 3].

c. Solution Segment

An all-artificial starting technique is used to begin the solution process. The initial basis consists of artificial arcs between each node and the root node. The flow on these arcs is set equal to the demand or supply requirements of the respective node. Arcs are oriented from the root to demand nodes and to the root from supply nodes. Assigning a relatively large cost to each artificial arc ensures that a feasible basis will contain no artificial arcs at positive flow. Artificial arcs present at the conclusion of the optimization with non-zero flow are explicitly identified.

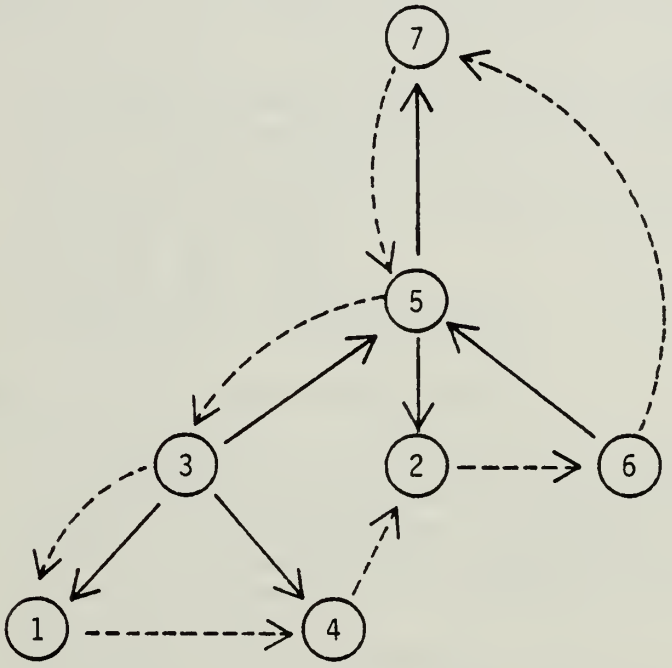
2. NLPNET

a. Input Data Requirements

NLPNET requires the input data illustrated in Figure 25. The source node, destination node, arc flow bounds, and initial flow are read directly into arc-length arrays. Since, in a nonlinear problem, all arcs potentially can have non-zero flow at optimality, the flow array

---

[5]As defined in the computer science literature [e.g., Ref. 36].



Node: i	1	2	3	4	5	6	7
Predecessor:	3	5	-5	3	7	-5	Δ
Traversal :	4	6	1	2	3	7	5
Depth :	3	2	2	3	1	2	0

---> Traversal

Fig. 23. GNET Basis Representation

Problem Definition (All Arc-Length Arrays Except as Noted)

H ( ) - A node-length array of entry points into arc-length arrays for each head node (e.g., all arcs directed toward head node i are found in locations H(i),..., H(i + 1) - 1 of arc length arrays).
T ( ) - Tail node indices (i.e., nodes which arcs are directed away from).
C ( ) - Cost per unit flow.
CP ( ) - Upper bound (capacity) on flow. (The sign bit is used to indicate a reflected arc.)

Basis Representation (All Node-length Arrays)

P ( ) - The predecessor of each node on its backpath to the root node. The orientation of the basic arc connecting node i to its predecessor is indicated by the sign of P(i), a negative value indicating an arc (i, - P(i)), and a positive value signifying an arc (P(i), i).
IT ( ) - Preorder traversal successors. IT(i) gives the node number of the next node to visit in preorder after node i. With P( ), IT( ) defines a basis triangulation and can be used for substitution solution.
D ( ) - Depths of the nodes in the current basis. D(i) gives the length of the backpath from node i to the root node (used in column generation).

Solution Representation (All Node-length Arrays)

X ( ),- Current flow of each basic arc, and capacity minus
CPX ( ) current flow of each basic arc (i.e., if P(i) = j < 0, flow is X(i)).

Fig. 24. GNET Data Structure

ARCS:

Source node number
Destination node number
Lowerbound on flow
Initial flow
Upperbound on flow
Cost function type
Cost function coefficients

NODES:

Net flow

Fig. 25. NLPNET Input Data Requirements

must be an arc-length structure. The bounds and the node pairs associated with each arc are explicitly maintained, in contrast to the GNET scheme where non-basic arcs are coded with a single bit to indicate status (at upper or lower bounds).

Nonlinear cost function definitions reside in the SYSTEM LIBRARY where they can easily be modified without requiring changes to any of the programs comprising the package. Each function is assigned a unique number upon inclusion in the library, so determination of a function's presence is a trivial matter. This is accomplished by checking function identification numbers against a master list located in the SYSTEM LIBRARY. Also required in the function definition are analytical expressions for the gradient and hessian. Functions can be defined using up to three coefficient terms; if more than three such coefficients are needed, then a small coding change in NLPNET is required.

Input node information is the same as for GNET (net flow only for non-default nodes). As in GNET, total supply and demand

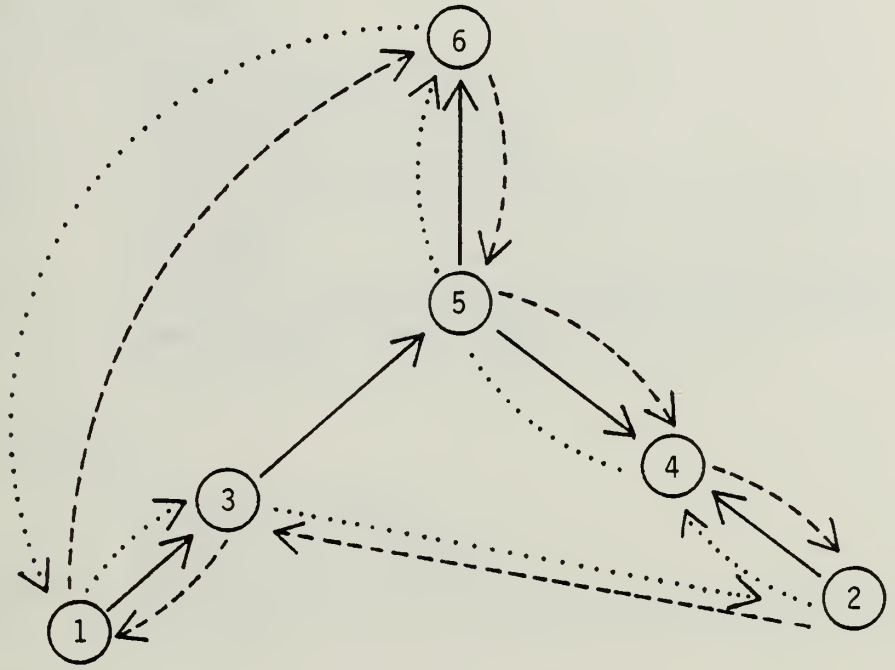
conservation must be achieved or the program terminates at the input stage.

b. Internal Data Structure

NLPNET also uses a root node to complete the basis. The basis tree is depicted by the familiar predecessor, depth, and traversal arrays which are functionally equivalent to the corresponding GNET structures. A fourth array, a reverse-traversal array, allows mobility opposite to that provided by the traversal array. This array is the inverse of the preorder traversal and is used for back-substitution during the calculation of the basic variable search direction induced by a superbasic direction. The basis representation is described by Figure 26. Arrays to maintain the variable partition, gradient vector, hessian diagonal vector, search direction, and dual variable estimates complete the data structure. These arrays allow NLPNET to perform primal simplex procedures directly on the basis representation in the spirit of GNET. A summary of the data structure is shown in Figure 27. A complete description of the arrays and their use is given by Dembo [Ref. 19].

c. User Definable Parameters

Real arithmetic is used extensively throughout NLPNET so various tolerances are necessary. Additionally, the solution process employs several parameters that control the performance, convergence, and accuracy characteristics of the solution process. Default settings for these values are built into the program, however, the user may redefine these settings (within established limits) if desired. Figure 28 is the menu that the user receives when the default settings are to be modified.



Node i	1	2	3	4	5	6
Predecessor*	3	4	5	-5	6	6
Traversal	6	3	1	2	4	5
Inverse Traversal	3	4	2	5	6	1
Depth	3	3	2	2	1	0

---> Traversal Path
····> Inverse Traversal Path
*Negative I = predecessor (J) implies that the arc is orientated from node I to node J.

Fig. 26. NLPNET Basis Representation

Problem Definition Arrays

IFROM ( ) - Source nodes of arcs.
ITO ( ) - Destination nodes of arcs.
LOWER ( ) - Lower bounds on arc flow.
UPPER ( ) - Upper bounds on arc flow.
ITYPE ( ) - Library identification numbers of arc objective functions.
COEF ( ) - Coefficients of objective function (0-3 per arc).
IPTC ( ) - Pointers to index of coefficient array contains first coefficient of each objective function.
RHS ( ) - Node net flow requirements

Variable Partition

LB ( ) - Basic arc indices.
LS ( ) - Superbasic arc indices.
LN ( ) - Nonbasic arc indices.
ISBEST( ) - Best superbasic variable to replace a given basic variable.

Basis Representation

LTHRD ( ) - Traversal array (thread).
LPRED ( ) - Predecessor array.
LRTHRD( ) - Inverse of traversal array (reverse thread).
NDEPTH( ) - Depth array.

Solution Representation

FLOW ( ) - Arc flow.
F ( ) - Arc contribution to objective function (at FLOW).
G ( ) - Gradient vector.
H ( ) - Hessian diagonal vector. (Hessian is a diagonal matrix.)
P ( ) - Search direction vector.
W ( ) - Dual variable estimates.

Fig. 27. NLPNET Data Structure

CURRENT PARAMETER VALUES

GENERAL TOLERANCES

1.	Alpha	= 0.0001
2.	Flow	= 0.0001
3.	Gradient	= 0.001
4.	Price-out	= 0.0001

LINESEARCH TOLERANCES

5.	Eps	= 0.001
6.	T	= 0.02
7.	Eta	= 0.1
8.	Epsd	= 0.001

OTHER

9.	Forcing ftn	= 1.0
10.	Max Iterations	= 50

Parameters OK? : Y(es or symbol to change -->

Fig. 28. NLPNET Default Modification Menu

d. Reports

The program prints iteration reports and final solution reports without user intervention. Either of these reports may be selectively disabled. In any event, the input data file's initial flow fields will always be updated upon normal program termination. Appendix B describes these reports.

3. ELASTICNET

This program combines both ENET and UNET into one composite solution module that is resident in memory for the duration of the optimization process. In this manner, costly disk access operations

resulting from UNET communication with ENET during "l-u" problem solution are eliminated. Of course, such a scheme requires more memory resources for code storage than would be the case with a partitioned format; however, UNET is a very concise code and the storage reduction is minimal.

The selection of the appropriate solution mechanism is automatically controlled by logic internal to ELASTICNET activated by the presence of "l-u" arcs in the input data file. Invocation of UNET in response to such arcs can be suppressed from the program option menu and only ENET utilized to process a file with "l-u" arcs (giving only a relaxed solution). This option allows a single file to represent both a free and "l-u" model.

a. Input Data Requirements

ELASTICNET exercises virtually the entire input data structure and extracts the data given in Figure 29 from the data file. For arcs, the source node, destination node, and unit cost are read into arrays identical to the GNET counterparts already described. ELASTICNET, unlike GNET, maintains explicit representations for each arc's upper and lower bounds on flow. This simplifies the program and results in a miniscule increase in storage requirements. Finally, the "l-u" status of each arc is maintained in an arc-length array with zeros indicating free arcs and ones marking "l-u" arcs.

Input node information consists of five numbers for each node: net flow at the node, upper and lower range on flow, and penalties for upper and lower range violation. In the absence of input information for ranges and penalties, default values are assigned by the input module. As with the other programs, all supply and demand nodes must

ARCS:

Source node number
Destination node number
Lowerbound on flow
Upperbound on flow
Cost information
"l-u" status

NODES:

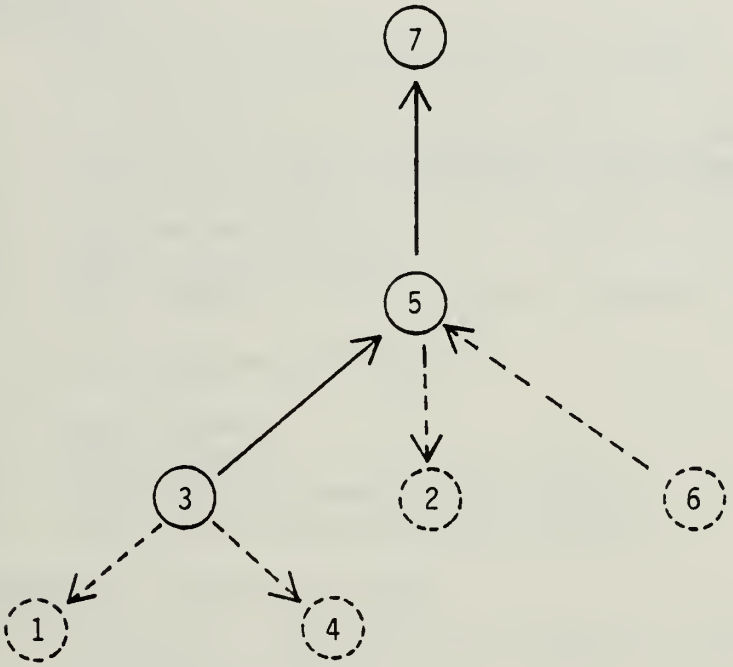
Net flow
Lower range on flow
Upper range on flow
Lower penalty for range violation
Upper penalty for range violation

Fig. 29. ELASTICNET Input Data Requirements

explicitly be present in the input file. Additionally, any transshipment nodes for which ranges or penalties differ from default values must be included. The magnitude of the default settings is user definable from the option menu. Flow conservation (supply = demand) is not required for the elastic model accommodated by ELASTICNET and therefore only a warning message is issued when supply does not equal demand.

b. Internal Data Structure

ELASTICNET maintains a basis representation (Figure 30) very similar to the GNET structure. Additional arrays are required to control the UNET enumeration process and record incumbent solutions. These structures are described in Figures 31 and 32.



Node: i	1	2	3	4	5	6	7
Predecessor:	3	5	-5	3	-7	-5	Δ
Traversal :	1	2	7	4	3	6	5
Depth :	0	0	3	0	2	0	1
Aggregate :	C_{13}	C_{25}	2	C_{34}	2	C_{56}	0

Fig. 30. ENET Basis Representation

Problem Definition

Arcs: (All Arc-Length Arrays Except as Noted)

H ( ) - A node-length array of entry points by head node into arc lists (sign bit indicates S_i reflected).
T ( ) - Tail node indices (sign bit indicates fixed arc).
C ( ) - Cost per unit flow.
BL ( ) - Lower bounds on flow.
BU ( ) - Upper bounds on flow (sign bit indicates reflected arc).

Nodes: (All Node-Length Arrays)

RL ( ) - Lower ranges.
RU ( ) - Upper ranges.
DL ( ) - Penalties for lower range violation.
DU ( ) - Penalties for upper range violation.

Basis Representation (Similar to GNET)

P ( ) - Predecessor array (sign bit indicates basic arc orientation).
IT ( ) - Traversal array.
D ( ) - Depth array.
A ( ) - Aggregated successor array (for an aggregated node, the cost is stored for its basic arc predecessor; for a disaggregated node, the number of aggregated successor nodes/basic arcs is stored) [Ref. 3, p. 28].

Solution Representation (All Node-Length Arrays)

X ( ),- Arrays with current flow of each basic arc, and
BUX ( ) capacity minus current flow of each basic arc (i.e., $P(i) = j < 0$, flow is $X(i)$).
U ( ) - Dual variables.

Fig. 31. ENET Data Structure

Enumeration Control

IFIX( ) - A last-in-first-out (LIFO) structure that records the arcs currently fixed at a bound. IFIX(i) is the arc number in the current enumeration of the i^{th} restriction.

LGB ( ) - Maintains (3-bit) "1-u" status of each arc. On input. "1-u" arcs are assigned a LGB value of one while all other arcs are given a value of zero. During enumeration, nonzero values of LGB indicate bounds at which arcs are fixed or reversed. Fixed arcs have negative T ( ) while free arcs have positive T ( ) values.

Incumbent Solution Record

IPS ( ) - The best primal solution encountered.

IDS ( ) - The dual solution corresponding to the best primal solution.

IB ( ) - Initial right hand side of constraints.

Fig. 32. UNET Data Structure

c. Model Alteration

Model alteration capability has been provided that allows the user to adjust arc and node parameters and resolve the modified problem. After viewing the new solution, the final problem attributes may be saved to a data file, the original file updated, or another adjustment cycle performed.

d. Reports

The program displays (on user designated peripherals) an echo print of the input file and final solution reports. Automatic generation of these reports is the default case, however, either report may be disabled. Appendix B contains examples of these reports.

V. COMPUTATIONAL RESULTS

This section describes the capabilities of the package with respect to problem size, solution speed, and versatility. As with most applied mathematical programming projects, availability of suitable example problems hindered the testing effort. Although standard test problems have been described in the literature, most are either very small academic examples or randomly-generated problems. The difficulty with using randomly-generated problems is that they are unstructured and therefore may not adequately exploit the efficiency of a programming code designed to solve "real-world" problems. It has been suggested [Ref. 3] that such random problems may even be harder to solve than naturally occurring formulations. Nevertheless, it has been necessary to include some randomly-generated problems to provide a complete assessment of package capabilities. All randomly generated problems were produced by NETGEN [Ref. 37] running on an IBM 370/3033 computer and subsequently transferred to the microcomputer using package file transfer features (EDITOR program and commercial data transfer program [Ref. 35]). Additionally, a few small academic examples were utilized. Finally, a preprocessor has been written to construct models which (although fictitious) more closely resemble real world situations than either of the two previously mentioned problem classes.

A. MAXIMUM PROBLEM SIZE

The maximum representable formulation (in terms of numbers of arcs and nodes) is a function of model type, the relative proportion of arcs

and nodes, and the microcomputer employed. All statements concerning problem size apply to the APPLE II microcomputer and no attempt has been made to translate the results to other hardware configurations.

Although one may specify a nominal ratio of arcs to nodes and configure the programs in that manner, it might be necessary to alter this proportion to accommodate a particular model. A limitation of this package is that in order to effect such an alteration, both the solution program pertaining to the class of models in question and the MASTER program must be recompiled. The coding change itself is trivial (two constants in each program control the partition), but the compilation process is time consuming. One solution to this dilemma would be to maintain multiple copies of the package, each with different problem size capabilities, to represent anticipated modeling requirements. Table 1 gives the partition employed during the development of the package.

The APPLE II microcomputer has approximately 39,900 words of memory available for program use after the Pascal operating system has been loaded. The combined code of the main segment of a solution program (always in memory) and that of its largest subordinate segment (typically the solution segment) will determine the memory available for data storage. In the absence of data partitioning, this also defines the maximum size of a representable model. The memory budget for each solution program is shown in Table 2.

Data storage memory requirements can be expressed as a function of the number of arcs and nodes represented and will, of course, differ for each solution program. These relationships are as follows:

TABLE 1

NOMINAL PACKAGE DATA PARTITION

	Nodes	Arcs
GNET	400	2000
ELASTICNET	300	500
NLPNET	50	175

TABLE 2

PACKAGE MEMORY USAGE

	Main Segment	Largest Auxiliary Segment	Available for Data
GNET	5072	7517	27311
ELASTICNET	6565	13898	19442
NLPNET	6804	19941	13155

Requirements are given in BYTES (8-bits) with 39,900 total BYTES available for program use.

1. GNET: $6 \mid A \mid + 20 \mid N \mid \leq 27311$,

2. ELASTICNET: $14 \mid A \mid + 26 \mid N \mid \leq 19442$,

3. NLPNET: $56 \mid A \mid + 22 \mid N \mid \leq 13155$,

where $\mid A \mid$ signifies the number of arcs and $\mid N \mid$ the number of nodes represented. A summary of this information for selected combinations of arcs and nodes is presented in Table 3.

B. SOLUTION TIMES

Solution times vary with the complexity of the model under consideration. The simplest formulation, inelastic linear models (GNET), requires only integer arithmetic for logical comparisions and therefore produces the fastest solution times for a given model size. Elastic models, although able to take advantage of some of the efficiencies associated with linear models, represent a versatile but complex formulation that necessitates additional work to cope with the increased information requirements. By specifying "1-u" arcs in a model, numerous subproblems (each equivalent to a single ENET run) must be solved and coordinated. The maximum number of such enumerations (worst case) grows exponentially with the number of "1-u" arcs ($2^{k+1} + 1$, where k is the number of "1-u" arcs), hence solution time increases proportionally. Nonlinear models require the most time to achieve optimality. These models employ vast amounts of real arithmetic (including transcendental function computations which are very time consuming on the APPLE II) to perform the necessary functional evaluations, direction finding and linesearch solutions. Nonlinear solution times are also highly sensitive to the form of the objective functions.

TABLE 3

ALLOWABLE ARC/NODE PARTITIONS

	ARCS		
NODES	GNET	ELASTICNET	NLPNET
20	3368	1351	227
30	3346	1333	223
40	3323	1314	219
50	3301	1295	215
60	3278	1277	211
70	3256	1258	207
80	3233	1240	203
90	3211	1221	199
100	3188	1203	185
150	3076	1110	175
200	2963	1017	
250	2851	924	
300	2738	831	
350	2626	738	
400	2513	645	
450	2401	553	
500	2288		
550	2176		
600	2063		
650	1951		
700	1838		
750	1726		
800	1613		
850	1501		
900	1388		
950	1287		
1000	1163		

A wide variety of test problems have been examined and the solution results are presented in Table 4. The solution times are all quite reasonable (although orders of magnitude slower than those obtained with large computers) and reflect the expected relationships between model classes. An example of the accuracy for a representative nonlinear formulation is given in Table 5.

C. MODELING FLEXIBILITY

To demonstrate the flexibility of the package, a preprocessor program has been written to model a three echelon product distribution network. In such a model, products flow from production centers to customers via distribution centers with storage or handling costs incurred at the intermediate echelon. The object of this formulation is to meet demands from available supplies at minimum transportation and handling cost. Transportation costs are modeled by arcs between components of the various model echelons. Handling costs can be depicted by adding an additional arc at each distribution center with appropriate cost function and flow bounds. Figure 33 shows the basic structure of such a model.

The preprocessor requires geographical coordinates and product flow requirements for each location to be modeled. All the transportation arcs are assigned linear cost functions proportional to the great circle distance between the various locations. Handling arc objective functions are assigned by the user to reflect the desired formulation. The user also specifies the minimum number of customers to be linked to each distribution center. The preprocessor then constructs a model as follows:

TABLE 4

TYPICAL SOLUTION TIMES

Problem	Nodes	Arcs	Solution† Time (Seconds)	Pivots or (Iterations)	Model‡ Class	Remarks
GNET1*	7	13	2	7	C	Multiple arcs
GNET2	15	50	7	41	C	
GNET3	40	100	21	76	C	
GNET4	100	375	117	197	A	
GNET5	200	990	276	314	U	
GNET6	200	990	280	338	C	
ENET1*	7	13	2	7	C	
ENET2	40	100	53	103	C	
ENET3	100	375	200	268	A	
UNET1*	7	13	13	23	C	1 "l-u" arc
UNET2*	7	13	12	24	C	2 "l-u" arcs
UNET3*	7	13	59	113	C	13 "l-u" arcs
NLPNET1*	5	8	40	(6)	C	mixed functions‡
NLPNET2*	7	12	14	(3)	C	all quadratic
NLPNET3*	13	18	164	(10)	C	mixed functions‡

* Indicates academic examples, all others are randomly generated problems.

† Elapsed time for solution module (exclusive of input/output operations).

‡ Hyperbolic SIN, power and linear functions.

‡ C - Capacitated transshipment,
† U - Uncapacitated transshipment,
A - Assignment.

TABLE 5

OPTIMAL SOLUTION COMPARISON

Arc	Optimal Solution Flow	
	APPLE II	DEC 20/60 (Double Precision)
1	52.1063	52.1063
2	47.8936	47.8937
3	18.5550	18.5552
4	33.5514	33.5511
5	26.4450	26.4448
6	21.4486	21.4489
7	5.000	5.000
8	6.60537	6.6050
9	6.9496	5.9502
10	14.6698	14.6613
11	10.0000	10.0000
12	8.88158	8.8898
13	0.0000	0.0000
14	13.3946	13.3950
15	13.0504	13.0498
16	0.33020	0.3387
17	0.0000	0.0000
18	21.1184	21.1102
Optimal Value of Objective Function	1453.420	1453.417

Objective function components are power, linear, and hyperbolic SIN functions. (Problem is NLPNET3).

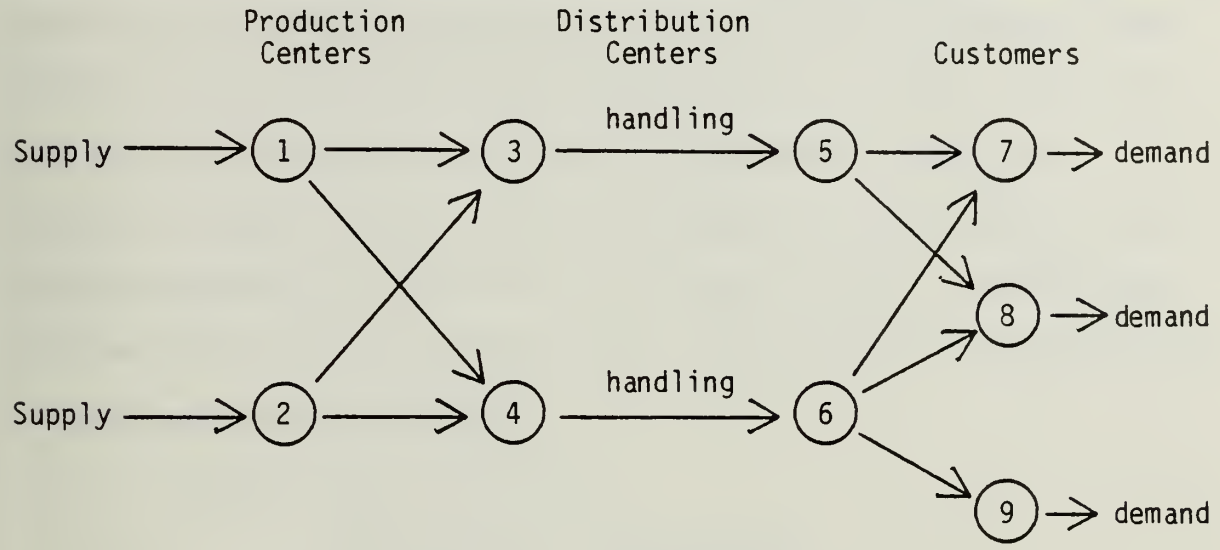


Fig. 33. Three Echelon Distribution Model With Handling Costs

STEP 1. Production centers are linked with each distribution center.

STEP 2. Handling arcs are added.

STEP 3. Each production center is connected with the required number of customers selected in order of proximity.

STEP 4. Any customer not linked to at least one distribution center by STEP 3 is connected to the closest distribution center.

The following types of handling cost functions can be accommodated:

1. Linear.
2. "1-u".
3. Fixed cost.
4. Nonlinear.

The linear formulation results, of course, in a completely linear model. Addition of "1-u" arcs transforms the model into one where distribution centers are either open at full capacity (with linear

handling costs) or closed. Fixed cost formulations incur an additional charge just for opening a distribution center (fixed cost) with a linear handling cost (variable cost) applied to each unit of flow. Such fixed cost models are generated using "1-u" arcs as shown in Figure 34. Finally, the handling costs may be represented by various nonlinear cost functions (potentially different for each handling arc). Two nonlinear functions that immediately come to mind as appropriate in a handling cost situation are quadratic functions:

$$\text{Cost} = f(x) = ax^2 + bx + c$$

$$\text{lower bound} \leq x \leq \text{upper bound},$$

and hybrid linear/quadratic functions:

$$\text{Cost} = f(x) = \begin{cases} cx & \text{lowerbound} \leq x \leq \text{changeover point}, \\ ax^2 + bx + (c)(\text{changeover point}) & \text{changeover point} < x \leq \text{upperbound}, \end{cases}$$

$$a \geq 0 \quad \text{(for convexity)}.$$

$$x \equiv \text{flow}$$

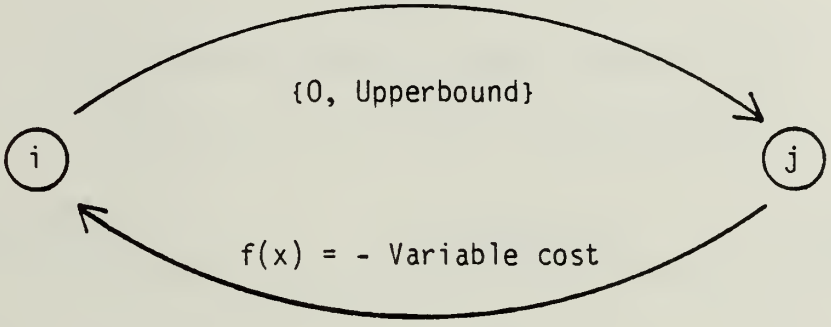
For demonstration purposes, a series of problems were generated with the following structural characteristics:

1. five production centers,
2. five distribution centers,
3. twenty-five customers, and
4. a minimum of eight customers per distribution center.

The resulting 40-node, 73-arc model was replicated using the various handling arc objective functions described above. Solution times for these test problems are given in Table 6.

Numerous modifications of the models are possible by exercising the "elastic" features of ENET and UNET. In this manner, realistic enhancements

$$f(x) = \frac{\text{fixed cost + variable cost x (upperbound-lowerbound)}}{\text{upperbound}}$$



0, upperbound-lowerbound

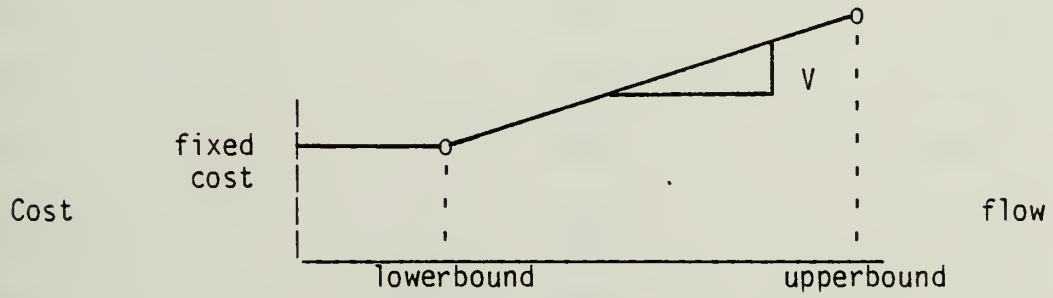


Fig. 34. Modeling Fixed Costs

such as optional surplus or shortage assignment can easily be incorporated into the model.

TABLE 6

THREE ECHELON DISTRIBUTION MODEL SOLUTION TIMES

(40 Nodes, 73 Arcs With 5 Handling Arcs)

Form of Handling Arc Objective Function	Solution Time (Seconds)	Pivots or (Iterations)
Linear (ENET)	40	92
One "1-u"	120	256
Two "1-u"	175	362
Five "1-u"	584	1179
One Fixed Cost	115	260
Two Fixed Cost	180	387
Five Fixed Cost	458	960
Quadratic	65	(4)
Hybrid Linear/Quadratic	68	(5)

VI. CONCLUSIONS AND RECOMMENDATIONS

The research effort described by this paper has been quite successful. Microcomputer adaptations of advanced algorithms for minimum cost network flow problems have been shown to be not only feasible but also practical for moderate-sized formulations. Additionally, an integrated repertoire of solution methods has been presented that illustrates the usefulness of such packages and serves as a prototype for their implementation on larger computer systems. This software fills a void in the existing spectrum of computer techniques for network problems by providing a single package to deal with a variety of diverse modeling situations.

Economic justification for the use of such microcomputer-based packages requires further investigation and could easily be the exclusive subject of a formal study. Certainly, the microcomputer will not replace larger computers, but instead will serve as a useful supplement. The capabilities of this software package infer that the microcomputer's niche lies in providing quick answers to relatively small problems. In this sense, a desktop computational device might be more convenient to use than the services of a large computer center. This is especially true if the software is user-friendly and easy to use.

There are certain situations where the microcomputer is clearly the only choice. Consider remote sites where access to large computers is limited or nonexistent. In such a scenario, the portability of the smaller computer provides the analyst with a powerful mathematical programming capability that would otherwise not be available. Indeed,

microcomputers have already been used in primitive locations utilizing rudimentary power supplies, so such a proposal is not idle conjecture.

The finite time horizons imposed on this project necessarily limited the scope of the initial study and there are many enhancements that could be pursued. First, the algorithms presented, although quite efficient, could be improved and further tuned for the microcomputer environment. Also, the use of partitioned data structures was not investigated as a means to solve larger problems with available resources. Successful use of such procedures could extend the usefulness of the package. Finally, the most obvious extension would be the inclusion of additional algorithms to accommodate other classes of network models. For example, a generalized network code recently made available to us by McBride [Ref. 38] would be a perfect complement to the algorithms already included.

It is hoped that the success of the work presented here will further stimulate the development of additional mathematical programming software for microcomputer use. As smaller and more capable computers are inevitable, the operations research community must be prepared to exploit their considerable potential.

APPENDIX A

PACKAGE EXTERNAL DATA STRUCTURE (DATA FILES)

```
CONST
  MXNUMCOEF     = 3;
  MXNAMELENGTH = 9;

TYPE
  SMALLSTRING   = STRING[MXNAMELENGTH];
  COEFARRAY     = ARRAY[1..MXNUMCOEF] OF REAL:
  RECTYPE       = (HEADER,ARC,NODE);
  NETWORK       = (LINEAR,NONLINEAR,ELASTIC,MIXEDINTEGER,GENERAL);
  NODETYPE      = (SUPPLY,DEMAND,TRANSSHIPMENT);
  NETRECORD     = RECORD
                    CASE ENTITY: RECTYPE OF
                      HEADER:  (PROBLEMNAME      : SMALLSTRING;
                                PROBLEMTYPE      : NETWORK;
                                NUMNODES         : INTEGER;
                                NUMARCS          : INTEGER;
                                DATECREATED      : SMALLSTRING;
                                DATELASTUPDATE   : SMALLSTRING);

                      ARC:     (ARCNAME          : SMALLSTRING;
                                SOURCENODE       : INTEGER;
                                DESTINATIONNODE  : INTEGER;
                                LOWERBOUND       : REAL;
                                UPPERBOUND       : REAL;
                                INITIALFLOW      : REAL;
                                CASE ARCTYPE: NETWORK OF
                                  LINEAR         : (UNITCOST : REAL;
                                                    ZEROUARC : BOOLEAN);
                                  NONLINEAR      : (TYPEFTN  : INTEGER;
                                                    NUMCOEF  : 1..MXNUMCOEF);
                                                    COEF: COEFARRAY);

                      NODE:    (NODENAME         : SMALLSTRING;
                                NODENUMBER       : INTEGER;
                                NODEKIND         : NODETYPE;
                                NETFLOW          : REAL;
                                LOWERRANGE       : REAL;
                                UPPERRANGE       : REAL;
                                LOWERPENALTY     : REAL;
                                UPPERPENALTY     : REAL);
                    END

  NETFILE : FILE OF NETRECORD:
```

APPENDIX B

TYPICAL REPORTS

This appendix contains three examples illustrating typical reports generated by the various solution programs. These examples are:

Example 1: Linearization of a small nonlinear formulation with an infeasible starting solution. This serves as an example of both MASTER program and GNET output.

Example 2: Typical NLPNET run of a feasible (starting solution) nonlinear formulation.

Example 3: Output from a small linear formulation solved with ELASTICNET.

```
                          APPLE-NET : SOLUTION MODULE
                          Version IIA   of  29 Aug 81
                          ==========================
DATE: 6 SEP 81

PROCESSING DATA:NLPNET1.NET, A NON LINEAR NETWORK PROGRAM.
NUMBER OF NODES = 5      NUMBER OF ARCS =  8
PROBLEM IS INFEASIBLE
LINEARIZATION PERFORMED:
7. Assignment of zero costs for all arcs.
     THETA (POINT) =      0.00000

SOLUTION MODULE CHAIN OF EVENTS:
GNET--> NLPNET--> SOLUTION...


                            APPLENET - GNET MODULE
                          VERSION IIIA OF 28 AUG 81
                            ======================
 2000 ARC, 400 NODE VERSION

FILE: DATA:NLPNET1.NET     CREATED: 1-SEP-81     UPDATED: 1-SEP-81
PROBLEM NAME IS NLPNET-1
NUMBER OF NODES = 5      NUMBER OF ARCS =  8

ARC LIST...

  ARC       FROM     TO      UNIT       LOWER       UPPER
  NAME      NODE    NODE     COST       BOUND       BOUND

     ONE       1       2      0.00        0.00       80.00
     TWO       1       3      0.00        0.00       60.00
   THREE       2       3      0.00        0.00       50.00
    FOUR       2       4      0.00        0.00       40.00
    FIVE       2       5      0.00        0.00       50.00
     SIX       3       4      0.00        0.00       20.00
   SEVEN       3       5      0.00        0.00       60.00
   EIGHT       5       4      0.00        0.00       60.00

NODE LIST ...     FOR THOSE NODES EXPLICITLY IN DATA FILE

  NODE       NODE        NET         NODE
  NAME      NUMBER       FLOW        TYPE

  SUP1         1        100.00     SUPPLY
UNAMED         2          0.00     TRANSSHIPMENT
UNAMED         3          0.00     TRANSSHIPMENT
  DEM1         4        -30.00     DEMAND
  DEM2         5        -70.00     DEMAND

SUPPLIES = 100      DEMAND = 100         INITIAL COST = 0.00


                                  FINAL SOLUTION
                                  ==============

     7 Pivots performed

     FROM      TO        FLOW        F(X )
     NODE     NODE

       1        2          80          0
       1        3          20          0
       2        4          30          0
       2        5          50          0
       3        5          20          0

     OPTIMAL COST =      0.00
```

Example 1

```
                                APPLENET - NLPNET MODULE
                                VERSION IIB of  5 SEP 81
                                ========================

 175 ARC, 50 NODE VERSION
DATE: 6 SEP 81

FILE: DATA:NLPNET1.NET          CREATED 1-SEP-81      UPDATED 1-SEP-81
PROBLEM NAME IS NLPNET-1
Number of Nodes = 5             Number of Arcs = 8

ARC LIST...
```

ARC NAME	FROM NODE	TO NODE	LOWER BOUND	INITIAL FLOW	UPPER BOUND	OBJECTIVE FUNCTION			
ONE	1	2	0.00	80.00	80.00	5	5.00	1.00	100.00
TWO	1	3	0.00	20.00	60.00	5	1.00	2.00	1000.0
THREE	2	3	0.00	0.00	50.00	0	25.00	0.00	
FOUR	2	4	0.00	30.00	40.00	6	1.00	0.50	2.50
FIVE	2	5	0.00	50.00	50.00	6	1.00	1.80	1.50
SIX	3	4	0.00	0.00	20.00	6	1.00	8.00	1.20
SEVEN	3	5	0.00	20.00	60.00	6	1.00	0.75	2.00
EIGHT	5	4	0.00	0.00	60.00	0	10.00	0.00	

```
NODE LIST...
```

NODE NAME	NODE NUMBER	NET FLOW	TYPE OF NODE
SUP1	1	100.00	SUPPLY
UNNAMED	2	0.00	TRANSSHIPMENT
UNNAMED	3	0.00	TRANSSHIPMENT
DEM1	4	-30.00	DEMAND
DEM2	5	-70.00	DEMAND

ITERATION NUMBER	OBJECTIVE VALUE	REDUCED GRADIENT	NUMBER OF FGH EVALNS	NUMBER OF CG ITERNS
0	1960.05	40.1657	1	0
1	1839.88	16.6275	2	1
2	1792.05	8.3739	3	3
3	1783.84	0.1302	4	4
4	1783.84	35.9362	5	5
5	1615.63	20.0252	8	6
6	1556.01	1.7029	9	7
7	1555.51	0.0166	10	8
8	1555.51	2.9732	16	9
9	1552.38	0.0583	17	10
10	1552.37	0.0000	18	12

Example 2, Page 1

FINAL SOLUTION

***** OPTIMAL ******

10 ITERATIONS
1.55237E3 * OBJECTIVE VALUE *

ARC NAME	FROM NODE	TO NODE	LOWER BOUND	FINAL FLOW	UPPER BOUND	F(x)	VARIABLE PARTITION
ONE	1	2	0.00	62.7304	80.000	6.69262	B
TWO	1	3	0.00	37.2696	60.000	3.76162	B
THREE	2	3	0.00	0.0000	50.000	0.00000	NB
FOUR	2	4	0.00	14.8383	40.000	149.930	B
FIVE	2	5	0.00	47.8920	50.000	800.393	SB
SIX	3	4	0.00	15.1617	20.000	316.668	SB
SEVEN	3	5	0.00	22.1080	60.000	274.929	B
EIGHT	5	4	0.00	0.0000	60.000	0.00000	NB

STATISTICS:

NUMBER OF PIVOTS = 1
NUMBER OF LINE SEARCHES (cubic) = 18
AVG SUPERBASIC ARCS/ITERATION = 1.2000

TOLERANCES:

GENERAL

ALPHA = 1.0000E-04
FLOW = 1.0000E-04
GRADIENT = 1.0000E-02

LINESEARCH

ETA = 1.00000E-01
EPS = 1.00000E-03
EPSD = 1.00000E-03
T = 1.00000E-03

APPLE-NET : SOLUTION MODULE
Version IIA of 29 Aug 81

DATE: 6 SEP 81

PROCESSING DATA:ENET1.NET, A LINEAR NETWORK PROGRAM.
NUMBER OF NODES = 7 NUMBER OF ARCS = 13

SOLUTION MODULE CHAIN OF EVENTS:
ELASTICNET--> SOLUTION...

APPLENET - ELASTIC/ "L-U" MODULE
VERSION IIA OF 3 SEP 81

DATE: 6 SEP 81

FILE: DATA:ENET1.NET CREATED: 6-SEP-81 UPDATED: 6-SEP-81
PROBLEM NAME IS ENET-1
NUMBER OF NODES = 7 NUMBER OF ARCS = 13

ARC LIST...

ARC NAME	FROM NODE	TO NODE	UNIT COST	LOWER BOUND	UPPER BOUND	"L-U" STATUS
AB	1	3	3.00	0.00	100.00	
AC	1	4	6.00	0.00	100.00	
AG	1	2	0.00	0.00	50.00	
BC	3	4	1.00	0.00	200.00	
BD	3	7	1.00	0.00	300.00	
CD	4	7	2.00	0.00	100.00	
CF	4	5	6.00	0.00	150.00	
ED	6	7	11.00	0.00	200.00	
FC	5	4	5.00	0.00	150.00	
FE	5	6	8.00	0.00	150.00	
FF	5	6	5.00	0.00	50.00	
GC	2	4	4.00	0.00	50.00	
GF	2	5	2.00	0.00	400.00	

NODE LIST ... FOR THOSE NODES EXPLICITLY IN DATA FILE

NODE NAME	NODE NUMBER	LOWER PENALTY	LOWER RANGE	NET FLOW	UPPER RANGE	UPPER PENALTY	NODE TYPE
-A	1	1000.00	90.00	90.00	90.00	1000.00	SUP
-C	2	1000.00	100.00	100.00	100.00	1000.00	SUP
D-	6	1000.00	-120.00	-120.00	-120.00	1000.00	DEM
E-	7	1000.00	-70.00	-70.00	-70.00	1000.00	DEM

MAX DUAL BOUNDS = 1000

FINAL SOLUTION

ARC ATTRIBUTES ...

ARC #	FROM NODE	TO NODE	UNIT COST	LOWER BOUND	FLOW	UPPER BOUND	REDUCED COST	F(X)
2	1	3	3	0	90	100	0	270
6	2	4	4	0	30	50	0	120
8	2	5	2	0	70	400	0	140
9	5	6	8	0	20	150	0	160
10	5	6	5	0	50	50	3	250
11	3	7	1	0	90	300	0	90
12	4	7	2	0	30	100	0	60

NODE ATTRIBUTES ...

NODE #	LOWER RANGE	NET FLOW	UPPER RANGE	LOWER DUAL	DUAL	UPPER DUAL
1	90	90	90	-1000	-4	1000
2	100	100	100	-1000	-6	1000
3	0	0	0	-1000	-1	1000
4	0	0	0	-1000	-2	1000
5	0	0	0	-1000	-4	1000
6	-70	-70	-70	-1000	4	1000
7	-120	-120	-120	-1000	0	1000

WOPT = 1.09000E3 OPT = 1.09000E3

OPTIMAL SOLUTION...

LIST OF REFERENCES

1. Bradley, S. P., Hax, A. C. and Magnanti, T. L., Applied Mathematical Programming, Addison-Wesley, 1977.

2. Bradley, G. H., "Survey of Deterministic Networks," AIIE Transactions, v. 7, pp. 222-234, September 1975.

3. Bradley, G. H., Brown, G. G. and Graves, G. W., "Design and Implementation of Large Scale Primal Transshipment Algorithms," Management Science, v. 1, pp. 1-34, September 1977.

4. Covvey, H. D. and McAlister N., Computer Consciousness: Surviving the Automated 80s, Addison-Wesley, 1980.

5. Beale, E. M. L., "The Blackett Memorial Lecture 1980. Operational Research and Computers: A Personal View," Journal of the Operational Research Society, v. 31, pp. 761-767, September 1980.

6. Tanenbaum, A. S., Computer Networks, Prentice-Hall, Inc., 1981.

7. Lee, D. M. and Fox, D. B, A Microcomputer Decision Analysis Support System (DASS), paper presented at CORS/ORSA/TIMS Joint Meeting, Toronto, Canada, 4 May 1981.

8. Visi-Calc[6], Personal Software Inc., Sunnyvale, California.

9. QuikDirt[7], Wyman Associates, San Mateo, California.

10. Morgeson, J. D., Statistics Programs for the Apple II Plus Computer, M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1981.

11. Isbell, R., A Statistical Analysis Package for the TRS-80 Microcomputer, M.S. Thesis, Naval Postgraduate School, Monterey, California, September 1981.

12. Elam, J., Klingman, D. and Mulvey, J., "An Evaluation of Mathematical Programming and Minicomputers," European Journal of Operational Research, v. 3, pp. 30-39, January 1979.

---

[6]Visi-Calc is a trademark of Personal Software Inc.

[7]QuikDirt is a trademark of Wyman Associates.

13. Wyman, F. P., "3000 Arcs for 3000 Bucks: How to Justify a Personal Computer for Your OR/MS Department," Interfaces, v. 9, pp. 75-80, August 1979.

14. McNelly, W. F., Letter to the Editor, Interfaces, v. 10, pp. 119-120, June 1980.

15. Kreitzberg, C. B. and Shneiderman, B., The Elements of FORTRAN Style, Harcourt Brace Jovanovich, 1972.

16. Thenson, A., Computer Methods In Operations Research, Academic Press, 1978.

17. Lientz, B. P., Computer Applications in Operations Research, Prentice-Hall, Inc., 1975.

18. Rowe, J. L., "Core of Apple Computer's Problem Seems Solved," Monterey Peninsula Herald, 3 July 1981, p. 15.

19. Dembo, R. S., "A Truncated-Newton Algorithm for Nonlinear Network Optimization," (in preparation).

20. Brown, G. G. and Graves, G. W., "Elastic Networks," Unpublished research notes.

21. Wirth, N., "The Programming Language Pascal," Acta Informatia, v. 1, pp. 35-63, 1971.

22. Shillington, K., "Structure: The Key to Pascal's Problem-Solving Power," Datamation, v. 25, pp. 151-152, July 1979.

23. Lewis, R. G., Pascal Programming for the APPLE, Reston Publishing Co., Inc., 1981.

24. Irvine, C. A., "UCSD Pascal System Makes Programs Portable," Electronic Design, v. 28, pp. 114-118, 14 August 1980.

25. Gagne, J., "An Introduction to Pascal," Microcomputing, n. 42, pp. 68-76, June 1980.

26. Kennington, J. L. and Helgason, R. V., Algorithms for Network Programming, John Wiley & Sons, 1980.

27. Dembo, R. S. and Steihaug, T, "Truncated-Newton Algorithms for Large Scale Unconstrained Optimization," School of Management, Yale University, Working Paper Number 48 (September 1980).

28. Dembo, R. S. and Klincewicz, J. C., "A Scaled Reduced Gradient Algorithm for Network Flow with Convex Separable Costs," School of Management, Yale University, Working Paper Number 21 (November 1979), (to appear in Mathematical Programming Studies).

29. Dembo, R. S., Eisenstat, S. C., and Steihaug, T., "Inexact Newton Methods," (to appear in SIAM Journal on Numerical Analysis).

30. Luenberger, D. G., Introduction to Linear and Nonlinear Programming, Addison-Wesley, 1973.

31. Dennis, J. E. and More, J. J., "Quasi-Newton Methods, Motivation and Theory," SIAM Review, V. 19, pp. 46-89, January 1977.

32. Murtagh, B. A. and Saunders, M. A., "Large Scale Linearly Constrained Optimization," Mathematical Programming, V. 14, pp. 41-72, 1978.

33. Murray, W. and Gill, P. E., "Safeguarded Steplength Algorithms for Optimization Using Descent Methods," National Physical Laboratory, NPL Report NAC 37 (August 1974).

34. Wirth, N., Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

35. Pascal Interactive Terminal Software, Software Sorcery, McLean, Va.

36. Knuth, D. E., The Art of Computer Programming Volume 1, Addison-Wesley, 1969.

37. Klingman, D., Napier, A., and Stutz, J. "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems, Management Science, V. 20, N. 5, pp. 814-821, January 1974.

38. McBride, R. D. "The Efficient Solution of Generalized Network Problems," Working Paper, Department of Finance and Business Economics, School of Business, University of Southern California, September 1981.

39. Brown, G. G. and Duff, R. H. "A Microcomputer Based Network Optimization Package." (Presentation and live demonstration). CORS/ORSA/TIMS Meeting, Toronto, Canada, 4 May 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	LIBRARY, CODE 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
4.	Professor Gerald G. Brown Code 55Bw Department of Operations Research Naval Postgraduate School Monterey, California 93940	36
5.	Professor Alan R. Washburn, Code 55Ws Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
6.	Major Richard H. Duff, USMC 9 Revere Road Monterey, California 93940	1

Thesis
D78365 Duff
c.1 194356

A microcomputer-based network optimization package.

8 MAY 86

27455
30574

Thesis
D78365 Duff
c.1 194356

A microcomputer-based network optimization package.

thesD78365
A microcomputer-based network optimizati
3 2768 000 98574 1
DUDLEY KNOX LIBRARY